O'REILLY®



图灵程序设计丛书



精通特征工程

Feature Engineering for Machine Learning

通过Python示例掌握特征工程基本原则和实际应用，
增强机器学习算法效果

[美] 爱丽丝·郑 阿曼达·卡萨丽 著

陈光欣 译



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS

译者介绍

陈光欣

毕业于清华大学并留校工作，主要兴趣为数据分析与数据挖掘。

数字版权声明


图灵程序设计丛书

精通特征工程

Feature Engineering for Machine Learning

[美] 爱丽丝·郑　阿曼达·卡萨丽 著
陈光欣 译

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®
O'Reilly Media, Inc.授权人民邮电出版社出版

人民邮电出版社
北　京

图书在版编目（CIP）数据

精通特征工程 / （美）爱丽丝·郑（Alice Zheng），
（美）阿曼达·卡萨丽（Amanda Casari）著；陈光欣译
. -- 北京：人民邮电出版社，2019.4
（图灵程序设计丛书）
ISBN 978-7-115-50968-0

Ⅰ. ①精… Ⅱ. ①爱… ②阿… ③陈… Ⅲ. ①机器学
习 Ⅳ. ①TP181

中国版本图书馆CIP数据核字(2019)第045692号

内 容 提 要

本书介绍大量特征工程技术，阐明特征工程的基本原则。主要内容包括：机器学习流程中的基本概念，数值型数据的基础特征工程，自然文本的特征工程，词频－逆文档频率，高效的分类变量编码技术，主成分分析，模型堆叠，图像处理，等等。

本书适合机器学习相关从业者和数据科学家阅读。

◆ 著　　[美] 爱丽丝·郑　阿曼达·卡萨丽
译　　陈光欣
责任编辑　岳新欣
责任印制　周昇亮

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　印刷

◆ 开本：800×1000　1/16
印张：10.75
字数：254千字　　　　2019年4月第1版
印数：1－3 500册　　　　2019年4月北京第1次印刷
著作权合同登记号　图字：01-2018-8085号

定价：59.00元
读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

版权声明

O’Reilly Media, Inc.介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 *Make* 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版、在线服务还是面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——*Wired*

“O’Reilly 凭借一系列非凡想法（真希望当初我也想到了）建立了数百万美元的业务。”

——*Business 2.0*

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——*CRN*

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——*Irish Times*

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野，并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去，Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——*Linux Journal*

目录

前言

简介

为了提取知识和做出预测，机器学习使用数学模型来拟合数据。这些模型将特征作为输入。**特征**就是原始数据某个方面的数值表示。在机器学习流程中，特征是数据和模型之间的纽带。**特征工程**是指从原始数据中提取特征并将其转换为适合机器学习模型的格式。它是机器学习流程中一个极其关键的环节，因为正确的特征可以减轻构建模型的难度，从而使机器学习流程输出更高质量的结果。机器学习从业者有一个共识，那就是建立机器学习流程的绝大部分时间都耗费在特征工程和数据清洗上。然而，尽管特征工程非常重要，专门讨论这个话题的著作却很少。究其原因，可能是正确的特征要视模型和数据的具体情况而定，而模型和数据千差万别，很难从各种项目中归纳出特征工程的实践原则。

然而，特征工程并不只是针对具体项目的行为，它有一些基本原则，而且最好结合具体情境进行解释说明。在本书中，每一章都集中阐述一个数据问题：如何表示文本数据或图像数据，如何为自动生成的特征降低维度，何时以及如何对特征进行标准化，等等。你可以将本书看作内容互有联系的短篇小说集，而不是一部长篇小说。每一章都对大量现有特征工程技术进行了简单介绍，它们综合在一起，阐明了特征工程的基本原则。

掌握一门学科不仅仅是要了解其中的定义以及能够推导公式。仅知道它的工作机制和用途是不够的，你还必须理解它为什么这样设计，它与其他技术有何联系，以及每种方法的优点和缺点。只有清楚地知道事情是如何完成的，对其中的基本原理有直观的理解，并能将知识融会贯通，才称得上精通。尽管一本好书可以让你初窥门径，但只靠读书不能登堂入室，你必须动手实践，将你的想法变成实际的应用，这是一个不断迭代的过程。在每次迭代中，我们都能将想法理解得更加透彻，并逐渐找到更巧妙、更有创造性的实现方法。本书的目的就是帮助你更好地实现想法。

本书力求追本溯源，数学方法倒在其次。我们不仅讨论**如何**去做，还尽力探究**为什么**这样

做。我们的目的是获得想法背后的**直觉**，这样就能知道何时以及如何去应用这些方法。每个人的学习方式各不相同，因此本书使用了大量的文字描述和图片来进行讲解。使用数学公式则是为了精确地表示直觉，并充当本书与其他资料之间的沟通桥梁。

本书的代码示例是用 Python 编写的，使用了大量免费和开源的程序包。NumPy 库提供了数值向量和矩阵操作；Pandas 提供了数据框，这是 Python 数据科学的基础数据结构；scikit-learn 是一个通用的机器学习包，包含了大量的模型和特征转换功能；Matplotlib 和样式库 Seaborn 提供了绘图和可视化支持。你可以在本书的 GitHub 仓库（https://github.com/alicezheng/feature-engineering-book）中找到 .ipynb 格式的示例。

前 4 章的节奏不快，因为要照顾一下那些刚刚接触数据科学和机器学习的读者。第 1 章介绍机器学习流程中的基本概念（数据、模型、特征等）。第 2 章研究数值型数据的基础特征工程：过滤、分箱、缩放、对数变换和幂次变换，以及交互特征。第 3 章开始介绍自然文本的特征工程，并研究词袋、*n*-gram 和短语检测等技术。第 4 章介绍 tf-idf（词频－逆文档频率），并将其作为特征缩放的一个例子，说明特征缩放为什么会有效。从第 5 章开始，节奏开始加快，我们要讨论高效的分类变量编码技术，包括特征散列化和分箱计数。第 6 章介绍主成分分析（PCA），此时我们已经深入到机器学习的腹地了。第 7 章将 *k*-均值聚类作为一种特征化技术，说明了模型堆叠这一重要概念。第 8 章专门讲解图像处理，图像数据的特征提取要比文本数据困难得多。我们先介绍两种手动提取特征的技术：SIFT 和 HOG，然后再介绍深度学习这种最新的图像特征提取技术。最后，第 9 章通过一个完整的例子（为一个学术论文数据集创建推荐器）演示几种技术的实际应用。

特征工程是一个非常宽泛的话题，每天都有新的方法被发明出来，尤其是在自动特征学习领域。为了控制本书的篇幅，我们必须做一些取舍。本书不讨论音频数据的傅里叶分析，尽管这是一个与线性代数中的特征分析（第 4 章和第 6 章会介绍）联系得非常紧密的美妙主题。我们也不讨论随机特征，它与傅里叶分析密切相关。通过讲解图像数据的深度学习，本书对特征学习做了简要的介绍，但没有深入介绍目前正在蓬勃发展的大量深度学习模型。还有一些高级研究方法也不在本书讨论范围内，比如随机投影、复杂的文本特征化模型（如词向量和布朗聚类）、隐含空间模型（如隐含狄利克雷分布和矩阵分解）。如果这些名词对你来说根本无所谓，那么恭喜你；如果你的兴趣所在是特征学习的最前沿，那本书可能不适合你。

本书假定你具有机器学习的基础知识，比如知道什么是模型、什么是向量，但我们会通过一个简单的复习使所有读者处于同一起点。如果你具有线性代数、概率分布和最优化方面的经验，那将会很有帮助，但这些经验不是必需的。

排版约定

本书使用了下列排版约定。

- 黑体字
  表示新术语和重点强调的内容。
- 等宽字体（`constant width`）
  表示程序片段，以及正文中出现的程序元素，比如变量、函数名、数据库、数据类型、环境变量、语句和关键字等。
- 加粗等宽字体（**`constant width bold`**）
  表示应该由用户输入的命令或其他文本。
- 等宽斜体（*`constant width italic`*）
  表示应该由用户输入的值或根据上下文确定的值替换的文本。

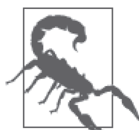
本书还包含一些线性代数公式。我们使用以下惯例：斜体小写字母表示标量（如 a），加粗斜体小写字母表示向量（如 $\boldsymbol{v}$），加粗斜体大写字母表示矩阵（如 $\boldsymbol{U}$）。



该图标表示提示或建议。



该图标表示一般注记。



该图标表示警告或警示。

使用示例代码

本书的附加资料（示例代码、练习题等）可以从 https://github.com/alicezheng/feature-engineering-book 下载。

本书是要帮你完成工作的。一般来说，如果本书提供了示例代码，你可以把它用在你的程序或文档中。除非你使用了很大一部分代码，否则无须联系我们获得许可。比如，用本书的几个代码片段写一个程序就无须获得许可，销售或分发 O’Reilly 图书的示例光盘则需要获得许可；引用本书中的示例代码回答问题无须获得许可，将书中大量的代码放到你的产品文档中则需要获得许可。

我们很希望但并不强制要求你在引用本书内容时加上引用说明。引用说明一般包括书名、作者、出版社和 ISBN。比如："*Feature Engineering for Maching Learning* by Alice Zheng and Amanda Casari (O'Reilly). Copyright 2018 Alice Zheng and Amanda Casari, 978-1-491-95324-2."

如果你觉得自己对示例代码的使用超出了上述许可的范围，欢迎你通过 permissions@oreilly.com 与我们联系。

O'Reilly Safari

Safari

Safari（原来叫 Safari Books Online）是一个会员制的培训和参考平台，面向企业、政府、教育从业者和个人。

会员可以访问几千种图书、培训视频、学习路径、互动式教程和精选播放列表，提供这些资源的出版商超过 250 家，包括 O'Reilly Media、Harvard Business Review、Prentice Hall Professional、Addison-Wesley Professional、Microsoft Press、Sams、Que、Peachpit Press、Adobe、Focal Press、Cisco Press、John Wiley & Sons、Syngress、Morgan Kaufmann、IBM Redbooks、Packt、Adobe Press、FT Press、Apress、Manning、New Riders、McGraw-Hill、Jones & Bartlett、Course Technology，等等。

要获得更多信息，请访问 http://oreilly.com/safari。

联系我们

请把对本书的评价和问题发给出版社。

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街 2 号成铭大厦 C 座 807 室（100035）
奥莱利技术咨询（北京）有限公司

本书有一个网页，上面列出了勘误[1]、示例和所有附加信息，网页地址为：
http://bit.ly/featureEngineering_for_ML

注 1：本书中文版勘误请到 www.ituring.com.cn/book/2050 查看和提交。——编者注

对于本书的评论和技术性问题，请发送电子邮件到：

bookquestions@oreilly.com

要了解更多 O’Reilly 图书、培训课程、会议和新闻的信息，请访问以下网站：

http://www.oreilly.com

我们在 Facebook 的地址如下：

http://facebook.com/oreilly

请关注我们的 Twitter 动态：

http://twitter.com/oreillymedia

我们的 YouTube 视频地址如下：

http://www.youtube.com/oreillymedia

致谢

首先，最应该感谢的是本书的编辑 Shannon Cutt 和 Jeff Bleiel，他们在本书漫长的出版过程（对此我们并不了解）中给予了两位初出茅庐的作者细致的指导。没有他们的努力工作，本书根本不会面世。还要感谢 O’Reilly 的图书策划编辑 Ben Lorica，是他的鼓励和肯定将本书从一个不切实际的想法变成了现实。感谢 Kristen Brown 和 O’Reilly 的制作团队，感谢他们对细节的极度关注和等待我们回应时的超级耐心。

对于数据科学家来说，出版一本书要比养个孩子付出更多的辛劳。对于所有帮助提高本书质量或使内容更加清晰的建议，我们都非常感激。Andreas Müller、Sethu Raman 和 Antoine Atallah 在百忙之中抽出宝贵的时间，帮我们进行了技术审阅。Antoine 不但神速地提供了反馈，还允许我们使用他强大的计算机来做实验。Ted Dunning 在统计学和应用机器学习方面的技术真是炉火纯青，他还极其慷慨地分享了自己的时间和想法，*k*-均值那一章中的方法和示例就是他贡献出来的。Owen Zhang 公开了他在 Kaggle 竞赛中使用响应速率特征的一些心得，并被 Misha Bilenko 收集进了机器学习的分箱计数方法集中。最后，还要感谢 Alex Ott、Francisco Martin 和 David Garrison 的反馈意见。

Alice的特别感谢

我要感谢 Graph/Data/Turi 团队在本书第一阶段中提供的大力支持。在与用户的交流中，我们产生了写作本书的想法。在为数据科学家建立一个新机器学习平台的过程中，我们发现人们需要更加系统化地理解特征工程。感谢 Carlos Guestrin，他允许我从繁忙的创业工作中抽身，集中精力进行写作。

感谢 Amanda，她先是作为技术审阅人，后来加入到了本书的写作过程中。你是最棒的终结者！本书已经完成，如果还想一边喝着茶和咖啡、吃着三明治和外卖，一边进行写作的话，我们就得找个新项目了。

特别感谢 Daisy Thompson，她是我的医生，也是我的朋友，感谢她在本书写作过程中给予我始终如一的支持。没有你的帮助，我需要更多时间才能下定决心，甚至会半途而废。就像在所有工作中一样，你给我的写作带来了光明与希望。

Amanda的特别感谢

因为只是写了一本书，不是获得了终身成就奖，所以我尽量只感谢那些与本书相关的人。

非常感谢 Alice 让我担任本书的技术编辑和合著者。从你的身上我学到了很多，包括如何写出好笑的数学笑话，以及如何将复杂概念解释清楚。

最后要特别感谢我的丈夫 Matthew，他尽了最大的努力来支持我、鼓励我向目标前进，从不让我蒙混过关。你是我最好的搭档，也是我最喜爱的伙伴。哪怕我只取得了一点点成就，你也以此为荣，并不断激励我。

电子书

扫描如下二维码，即可购买本书电子版。



第 1 章

机器学习流程

在开始学习特征工程之前，我们先花点时间了解一下机器学习的整体流程，这有助于我们更全面地理解特征工程。为此，我们先学习几个基本概念，比如**数据**和**模型**。

1.1 数据

我们所说的**数据**是对现实世界的现象的观测。例如，股票市场数据包括对每日股价、各个公司的盈余公告，甚至专家学者发表的股评文章的观测；个人的生物特征数据包括对每分钟的心率、血糖水平、血压等指标的测量；客户智能数据包括像“爱丽丝在星期天买了两本书”“鲍勃在这个网站上浏览了这些网页”“查理点击了这个从上周开始的特价销售链接”之类的数据。不同领域的数据示例无穷无尽，不一而足。

每份数据都是管中窥豹，只能反映一小部分现实，把这些观测综合起来才能得到一个完整的描述。但这个描述非常散乱，因为它由成千上万个小片段组成，而且总是存在测量噪声和缺失值。

1.2 任务

我们为什么要收集数据呢？因为有些问题需要靠数据找出答案，比如：“我应该买哪支股票？”“我如何活得更健康？”“我如何才能了解顾客不断变化的喜好，从而更好地提供服务？”

从数据到答案的路上，充满了错误的开始和死胡同（见图 1-1），经常是有意栽花花不发，

无心插柳柳成荫。数据处理工作流往往是多阶段的迭代过程。举个例子，股票价格是在交易所中观测到的，然后由像汤森路透这样的中间机构进行汇集并保存在数据库中，之后被某个公司买去，转换为一个 Hadoop 集群上的 Hive 仓库，再被某个脚本从仓库中读出，进行二次抽样和各种处理，接着通过另一个脚本进行清洗，导出到一个文件，转换为某种格式，然后你使用 R、Python 或 Scala 中你最喜欢的建模程序进行试验。接着，预测结果被导出为一个 CSV 文件，再用一个估值程序进行解析。模型会被迭代多次，由产品团队用 C++ 或 Java 重写，并在全部数据上运行，然后最终的预测结果会输出到另一个数据库中保存起来。

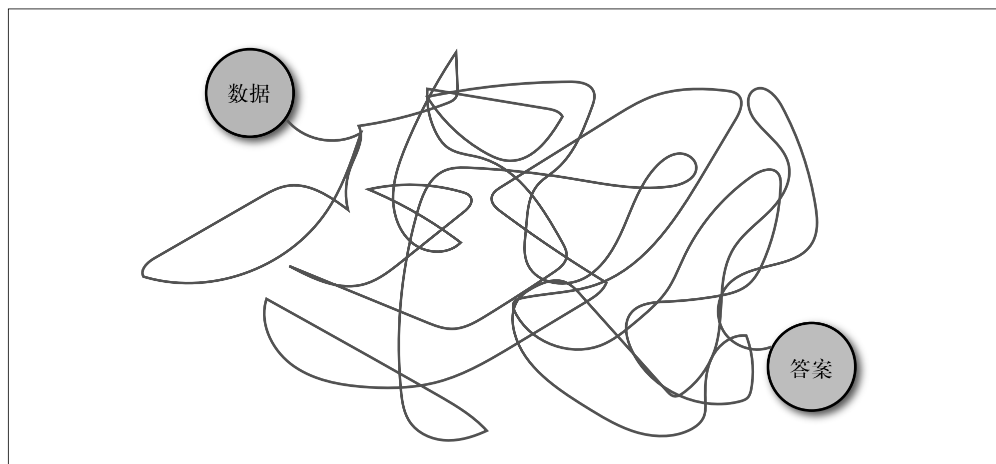


图 1-1：数据与答案之间错综复杂的路径

然而，如果我们不被这些杂乱的工具与系统所迷惑，就能够发现这个过程包括两个构成机器学习基础的数学实体：**模型**和**特征**。

1.3 模型

通过数据来理解世界就像是玩拼图，但这副拼图是杂乱且不完整的，而且带有多余的部分。这时数学模型——特别是统计模型——就派上用场了。统计语言中有很多概念，可以描述常见的数据特性，比如**错误数据**、**冗余数据**和**缺失数据**。错误数据是由测量时的错误造成的。冗余数据则是对同一信息的多次表述，比如，一周中的一天可以用分类变量来表示，它的值为“星期一”“星期二”……“星期日”，还可以表示为 0 和 6 之间的整数值。如果某些数据点中不存在这种星期几的信息，那就出现了缺失数据。

数据的**数学模型**描述了数据不同部分之间的关系。例如，预测股票价格的模型可以是一个公式，它将公司的收入历史、过去的股票价格和行业映射为预测的股票价格。音乐推荐模型可以基于收听习惯测量用户之间的相似度，然后向收听大量同种歌曲的用户推荐同一个音乐家。

数学公式将数值型的变量联系起来，但原始数据经常不是数值型的。（“爱丽丝在星期三购买了《指环王》三部曲”这一行为就不是数值型的，她随后对这本书发表的评价也不是数值型的。）必须有个什么东西将这二者联系起来，这就是特征的用武之地了。

1.4　特征

特征是原始数据的数值表示。有多种方法可以将原始数据转换为数值型的表示，所以特征可以有多种形式。当然，特征必须采用可用的数据类型。事实上，特征还和模型相关联，这一点可能并不那么显而易见。有些模型更适合使用某种类型的特征，反之亦然。正确的特征应该适合当前的任务，并易于被模型所使用。**特征工程**就是在给定数据、模型和任务的情况下设计出最合适的特征的过程。

特征的数量也非常重要。如果没有足够的有信息量的特征，那么模型将不能完成最终的任务。如果特征过多，或者多数特征不合适，那么模型将很难训练而且训练成本高昂。在训练过程中可能会出现一些影响模型性能的错误。

1.5　模型评价

特征和模型位于原始数据和我们想得到的知识之间（见图 1-2）。在机器学习流程中，我们要选择的不仅是模型，还有特征。模型与特征相辅相成，对其中一个的选择会影响另一个。好的特征可以使随后的建模步骤更容易，最后得出的模型也更能完成所需的任务。坏的特征要想达到同等性能，则需要复杂得多的模型。在本书后面的内容中，我们将介绍各种不同类型的特征，并讨论它们对于不同类型的数据和模型的优缺点。闲话少说，我们开始吧！

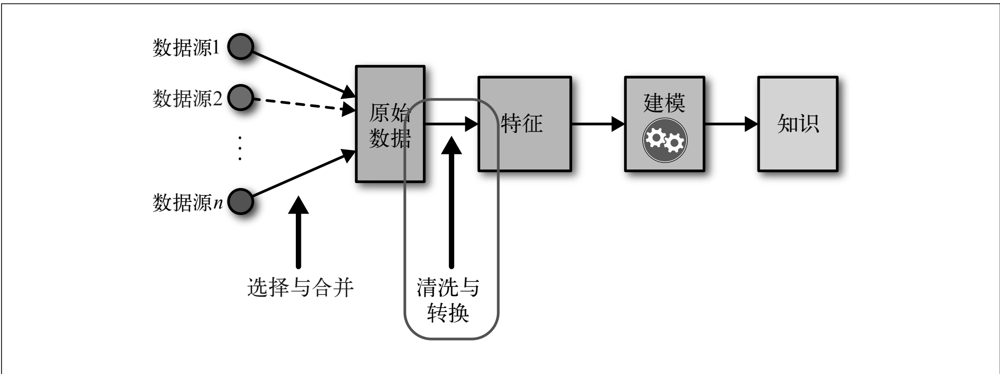


图 1-2：特征工程在机器学习流程中的位置

第 2 章

简单而又奇妙的数值

在研究诸如文本和图像这样的复杂数据类型之前，我们先看看最简单的数据类型：数值。数值型数据有多种来源：某个人或某个地点的地理位置、一宗交易的价格、传感器中的测量数据、交通流量，等等。尽管数值型数据已经很容易被数学模型所使用了，但并不意味着不需要进行特征工程。好的特征不仅能够表示出数据的主要特点，还应该符合模型的假设，因此通常必须进行数据转换。数值型数据的特征工程技术是非常基本的，只要原始数据被转换成数值型特征，就可以应用这些技术。

要对数值型数据进行合理性检查，首先要看看它的量级。我们只需知道它是正的还是负的，还是只需在一个很粗的粒度上知道它的量级？这种合理性检查非常重要，尤其是对于那些自动产生的数值，比如计数（网站的每日访问量、餐馆的评价数量等）。

然后，还要考虑一下特征的尺度。它的最大值和最小值是多少？是否横跨多个数量级？如果模型是输入特征的平滑函数，那么它对输入的尺度是非常敏感的。例如，$3x + 1$ 是输入 x 的一个简单线性函数，它的输出尺度直接取决于输入尺度。此外，*k*-均值聚类、最近邻方法、径向基核函数，以及所有使用欧氏距离的方法都属于这种情况。对于这类模型和模型成分，通常需要对特征进行**标准化**，以便将输出控制在期望的范围之内。

相反，逻辑函数对输入特征的尺度并不敏感。无论输入如何，这种函数的输出总是一个二值变量。例如，逻辑操作 AND 接受两个变量，当且仅当这两个输入都为真时，才输出 1。逻辑函数的另一个例子是阶梯函数（如：输入 x 是否大于 5？）。决策树模型中使用了输入特征的阶梯函数，因此，基于空间分割树的模型（决策树、梯度提升机、随机森林）对尺度是不敏感的。唯一的例外是，如果输入的尺度是随时间变化的，也就是说如

果特征是某种累计值，那么它最终可能会超出训练树的取值范围。如果出现了这种情况，就必须定期地对输入尺度进行调整；另外一种解决方案是使用区间计数方法，我们将在第 5 章中介绍。

数值型特征的分布也非常重要，这种分布可以表示出一个特定值出现的概率。输入特征的分布对于某些模型来说比其他模型更重要。例如，线性回归模型的训练过程需要假定预测误差近似地服从高斯分布。这种假定通常是没有问题的，除了预测目标分布在多个数量级中的时候，在这种情况下，误差符合高斯分布的假定将不会被满足。一种解决方法是对输出目标进行转换，以消除数量级带来的影响。（严格说来，这应该称为目标工程，而不是特征工程。）对数变换（**指数变换**的一种特殊形式）可以使变量的分布更加接近于高斯分布。

除了对特征进行转换以满足模型或训练过程中的假设，还可以将多个特征组合在一起形成更复杂的特征。我们希望复杂特征能更加简洁地捕获到原始数据中的重要信息。使输入特征“更具信息量”可以使模型本身更简单，更容易训练和评价，也能做出更好的预测。极端情况下，我们可以使用统计模型的输出作为复杂特征，这种思想称为**模型堆叠**，我们将在第 7 章和第 8 章中详细介绍。本章只给出一个最简单的复杂特征示例：**交互特征**。

交互特征的构造非常简单，但将特征组合输入到模型中后，会产生各种各样的结果。为了降低计算成本，通常需要使用自动的**特征选择**技术对输入特征进行精简。

我们先介绍一些基本概念：标量、向量和空间，然后讨论特征尺度、特征分布、交互特征和特征选择。

2.1 标量、向量和空间

在深入介绍之前，我们需要定义几个基本概念，作为后续内容的基础。单独的数值型特征称为**标量**，标量的有序列表称为**向量**，向量位于**向量空间**中。在绝大多数机器学习应用中，模型的输入通常表示为数值向量。本书后面会介绍将原始数据转换为数值向量的最佳实践策略。

向量可以形象化地表示为空间中的一个点。（有时候，人们会从原点向这个点画一条线或一个箭头。在本书中，我们通常只画出一个点。）举个例子，假如有一个二维向量 $\boldsymbol{v} = [1, -1]$，这个向量包含两个数值：在第一个方向 d_1 中，向量有一个值 1，在第二个方向 d_2 中，向量有一个值 −1。我们可以在二维图表中绘制出 $\boldsymbol{v}$，如图 2-1 所示。

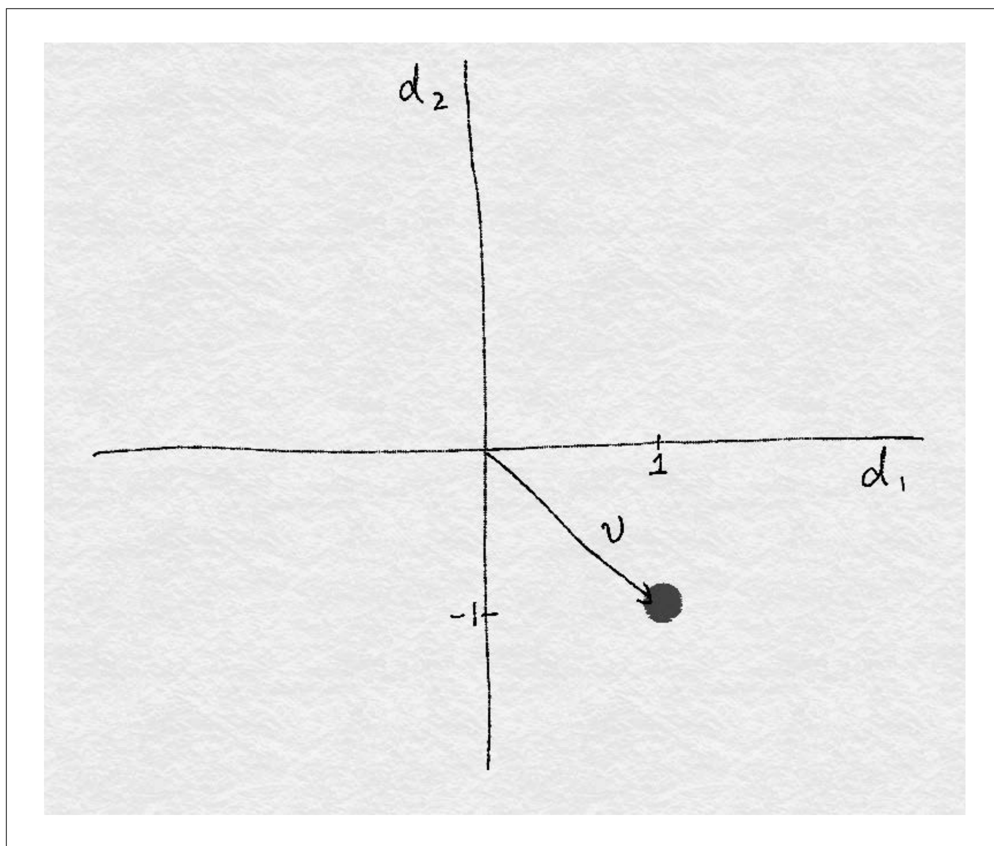


图 2-1：一个向量

在数据世界中，抽象的向量和它的特征维度具有实际意义。例如，可以用向量表示某个人对歌曲的偏好，这时每首歌都是一个特征，值为 1 时表示喜欢这首歌，值为 −1 时表示不喜欢。假设向量 ***v*** 表示听众 Bob 的偏好，他喜欢 Bob Dylan 的“Blowin’in the Wind”和 Lady Gaga 的“Poker Face”。其他听众则会有不同的偏好。总体来说，所有数据的集合可以在**特征空间**中形象地表示为一个点云。

反之，一首歌曲也可以通过一组听众的个人偏好表示出来。假设只有两个听众，Alice 和 Bob。Alice 喜欢“Poker Face”“Blowin’in the Wind” 和 Leonard Cohen 的“Hallelujah”，但是讨厌 Katy Perry 的“Roar” 和 Radiohead 的“Creep”。Bob 喜欢“Roar”“Hallelujah”和“Blowin’in the Wind”，但是讨厌“Poker Face”和“Creep”。每首歌都是听众空间中的一个点。就像可以在特征空间中表示出数据一样，我们也可以在**数据空间**中表示出特征。图 2-2 中展示出了这个示例。

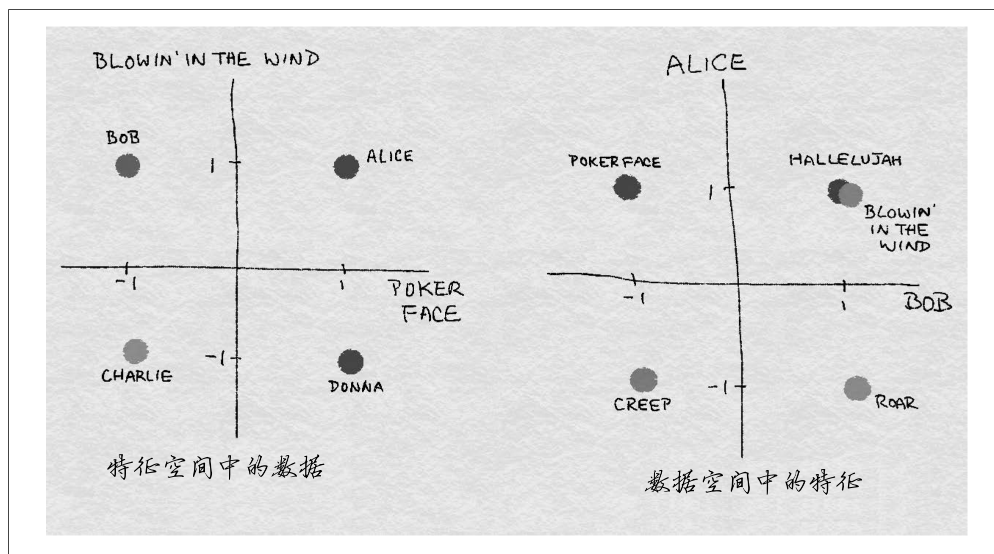


图 2-2：特征空间与数据空间

2.2 处理计数

在大数据时代，计数数据可以无限度地快速增长。用户可以无数次地重复播放一首歌曲或一部电影，也可以使用脚本反复地检查一个热门演出是否有余票，这都可以使播放计数和网站访问计数快速增长。当数据被大量且快速地生成时，很有可能包含一些极端值。这时就应该检查数据的尺度，确定是应该保留数据原始的数值形式，还是应该将它们转换成二值数据，或者进行粗粒度的分箱操作。为了说明这种思想，我们来看几个例子。

2.2.1 二值化

Echo Nest Taste Profile 是百万歌曲数据集（Million Song Dataset）的正式用户数据子集，其中包含了 100 万用户在 Echo Nest 网站上完整的音乐收听历史。下面是这个数据集的一些重要统计信息。

> **Echo Nest Taste Profile 数据集的统计信息**
>
> - 数据集中有超过 4800 万个由用户 ID、歌曲 ID 和收听次数构成的三元组。
> - 完整的数据集中包括 1 019 318 个独立用户和 384 546 首独立歌曲。

假设我们的任务是创建一个向用户推荐歌曲的推荐器，它的一个功能是预测某个用户喜欢某首歌曲的程度。既然数据中包含了实际的收听次数，那么可以用它作为预测的目标变量吗？如果高收听次数意味着用户真的喜欢这首歌，低收听次数意味着用户对这首歌不感兴

趣，那么就可以用它作为目标变量。但是，数据表明，尽管 99% 的收听次数是 24 或更低，还是有一些收听次数达到了几千，最大值是 9667。（如图 2-3 所示，直方图中的峰值出现在与 0 最接近的区间中，有 1 万多个三元组具有较高的收听次数，其中一些有几千次。）这些值高得离谱，如果我们试图去预测实际的收听次数，模型会被这些异常值严重带偏。

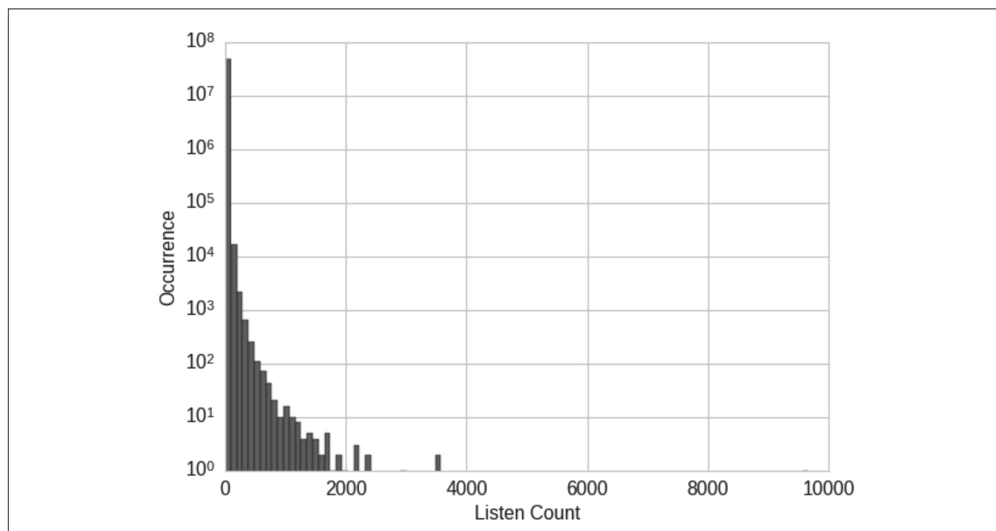


图 2-3：百万歌曲数据集 Taste Profile 子集中的收听次数直方图，注意 *y* 轴使用的是对数标度

在百万歌曲数据集中，原始的收听次数并不是衡量用户喜好的**强壮**指标。（在统计学术语中，“强壮”意味着该方法适用于各种情况。）不同的用户有不同的收听习惯，有些人会无限循环地播放他们最喜欢的歌曲，有些人则只是在特定情形下欣赏音乐。我们不能认为收听了某首歌曲 20 次的人喜欢该歌曲的程度肯定是收听了 10 次的人的两倍。

更强壮的用户偏好表示方法是将收听次数二值化，把所有大于 1 的次数值设为 1，如例 2-1 所示。换言之，如果用户收听了某首歌曲至少一次，那么就认为该用户喜欢该歌曲。这样，模型就不用花费开销来预测原始收听次数之前的时间差别。二值目标变量是一个既简单又强壮的用户偏好衡量指标。

例 2-1　百万歌曲数据集中的收听次数二值化

```
>>> import pandas as pd
>>> listen_count = pd.read_csv('millionsong/train_triplets.txt.zip',
...                            header=None, delimiter='\t')
# 表中包含有形式为“用户-歌曲-收听次数”的三元组。只包含非零收听次数。
# 因此，要二值化收听次数，只需将整个收听次数列设为1。
>>> listen_count[2] = 1
```

这是一个对模型的目标变量进行处理的例子。严格说来，目标变量不是特征，因为它不是输入。但为了正确地解决问题，有时候确实需要修改目标变量。

2.2.2　区间量化（分箱）

在这个练习中，我们使用来自于 Yelp 点评网站数据竞赛（http://www.yelp.com/dataset_challenge）第 6 轮的数据，并用这些数据创建一个规模小得多的分类数据集。Yelp 数据集是用户对商家的点评数据，这些商家来自于北美和欧洲的 10 个城市，每个商家都用 0 个或多个分类进行标记。

> **Yelp 点评数据集（第 6 轮）的统计信息**
>
> - 782 个商业分类。
> - 整个数据集包含 1 569 264（≈ 160 万）条点评和 61 184 个商家。
> - 就点评数量而言，“Restaurants”（990 627 条点评）和“Nightlife”（210 028 条点评）是最普遍的分类。
> - 没有商家既属于餐馆又属于夜生活场所，所以这两个点评分组中没有重叠。

每个商家都有一个点评数量。假设我们的任务是使用协同过滤方法预测某用户给某商家的打分。点评数量会是一个非常有用的输入特征，因为人气和高评分之间通常有很强的相关性。现在的问题就是，我们应该使用原始点评数量，还是应该对其做进一步的处理？图 2-4 是根据例 2-2 生成的，它给出了所有商家点评数量的直方图。从中可以看出，它和前一个例子中的收听次数具有相同的模式：大多数商家的点评数量很少，但有些商家具有几千条点评。

例 2-2　Yelp 数据集中的商家点评数量可视化

```
>>> import pandas as pd
>>> import json

# 加载商家数据
>>> biz_file = open('yelp_academic_dataset_business.json')
>>> biz_df = pd.DataFrame([json.loads(x) for x in biz_file.readlines()])
>>> biz_file.close()

>>> import matplotlib.pyplot as plt
>>> import seaborn as sns

# 绘制点评数量直方图
>>> sns.set_style('whitegrid')
>>> fig, ax = plt.subplots()
>>> biz_df['review_count'].hist(ax=ax, bins=100)
>>> ax.set_yscale('log')
>>> ax.tick_params(labelsize=14)
>>> ax.set_xlabel('Review Count', fontsize=14)
>>> ax.set_ylabel('Occurrence', fontsize=14)
```

原始的点评数量横跨了若干个数量级，这对很多模型来说都是个问题。在线性模型中，同一线性系数应该对所有可能的计数值起作用。过大的计数值对无监督学习方法也会造成破坏，比如 *k*-均值聚类，它使用欧氏距离作为相似度函数来测量数据点之间的相似度。

数据向量某个元素中过大的计数值对相似度的影响会远超其他元素，从而破坏整体的相似度测量。

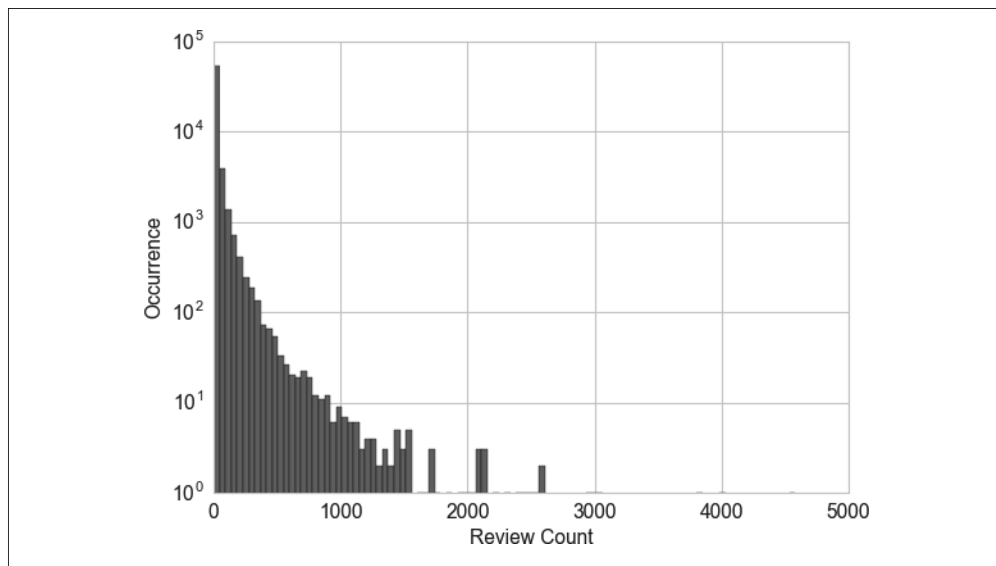


图 2-4：Yelp 点评数据集中商家点评数量直方图，*y* 轴使用对数标度

一种解决方法是对计数值进行**区间量化**，然后使用量化后的结果。换言之，我们将点评数量分到多个箱子里面，去掉实际的计数值。区间量化可以将连续型数值映射为离散型数值，我们可以将这种离散型数值看作一种有序的分箱序列，它表示的是对密度的测量。

为了对数据进行区间量化，必须确定每个分箱的宽度。有两种确定分箱宽度的方法：固定宽度分箱和自适应分箱。对于每种方法，我们都会给出一个示例。

1. 固定宽度分箱

通过固定宽度分箱，每个分箱中会包含一个具体范围内的数值。这些范围可以人工定制，也可以通过自动分段来生成，它们可以是线性的，也可以是指数性的。例如，我们可以按 10 年为一段来将人员划分到多个年龄范围中：0~9 岁的在分箱 1 中、10~19 岁的在分箱 2 中，等等。要将计数值映射到分箱，只需用计数值除以分箱的宽度，然后取整数部分。

我们也经常使用人工设计的年龄范围，它可以更好地反映出生命阶段，例如：

- 0~12 岁
- 12~17 岁
- 18~24 岁
- 25~34 岁
- 35~44 岁

- 45~54 岁
- 55~64 岁
- 65~74 岁
- 75 岁以上

当数值横跨多个数量级时，最好按照 10 的幂（或任何常数的幂）来进行分组：0~9、10~99、100~999、1000~9999，等等。这时分箱宽度是呈指数增长的，从 *O*(10) 到 *O*(100)、*O*(1000) 以及更大。要将计数值映射到分箱，需要取计数值的对数。指数宽度分箱与对数变换的关系非常紧密，我们将在 2.3 节中讨论。例 2-3 演示了几种分箱方法。

例 2-3　通过固定宽度分箱对计数值进行区间量化

```
>>> import numpy as np

# 生成20个随机整数，均匀分布在0~99之间
>>> small_counts = np.random.randint(0, 100, 20)
>>> small_counts
array([30, 64, 49, 26, 69, 23, 56,  7, 69, 67, 87, 14, 67, 33, 88, 77, 75,
       47, 44, 93])
# 通过除法映射到间隔均匀的分箱中，每个分箱的取值范围都是0~9
>>> np.floor_divide(small_counts, 10)
array([3, 6, 4, 2, 6, 2, 5, 0, 6, 6, 8, 1, 6, 3, 8, 7, 7, 4, 4, 9], dtype=int32)

# 横跨若干数量级的计数值数组
>>> large_counts = [296, 8286, 64011, 80, 3, 725, 867, 2215, 7689, 11495, 91897,
...                 44, 28, 7971, 926, 122, 22222]
# 通过对数函数映射到指数宽度分箱
>>> np.floor(np.log10(large_counts))
array([ 2.,  3.,  4.,  1.,  0.,  2.,  2.,  3.,  3.,  4.,  4.,  1.,  1.,
        3.,  2.,  2.,  4.])
```

2. 分位数分箱

固定宽度分箱非常容易计算，但如果计数值中有比较大的缺口，就会产生很多没有任何数据的空箱子。根据数据的分布特点，进行自适应的箱体定位，就可以解决这个问题。这种方法可以使用数据分布的分位数来实现。

分位数是可以将数据划分为相等的若干份数的值。例如，中位数（即二分位数）可以将数据划分为两半，其中一半数据点比中位数小，另一半数据点比中位数大。四分位数将数据四等分，十分位数将数据十等分，等等。例 2-4 演示了计算 Yelp 商家点评数量十分位数的方法。在图 2-5 中，我们将十分位数覆盖在直方图上，由此可以更加清楚地看出，数据是向较小的计数值偏斜的。

例 2-4　计算 Yelp 商家点评数量的十分位数

```
>>> deciles = biz_df['review_count'].quantile([.1, .2, .3, .4, .5, .6, .7, .8, .9])
>>> deciles
0.1     3.0
0.2     4.0
```

```
0.3     5.0
0.4     6.0
0.5     8.0
0.6    12.0
0.7    17.0
0.8    28.0
0.9    58.0
Name: review_count, dtype: float64

# 在直方图上画出十分位数
>>> sns.set_style('whitegrid')
>>> fig, ax = plt.subplots()
>>> biz_df['review_count'].hist(ax=ax, bins=100)
>>> for pos in deciles:
...     handle = plt.axvline(pos, color='r')
>>> ax.legend([handle], ['deciles'], fontsize=14)
>>> ax.set_yscale('log')
>>> ax.set_xscale('log')
>>> ax.tick_params(labelsize=14)
>>> ax.set_xlabel('Review Count', fontsize=14)
>>> ax.set_ylabel('Occurrence', fontsize=14)
```

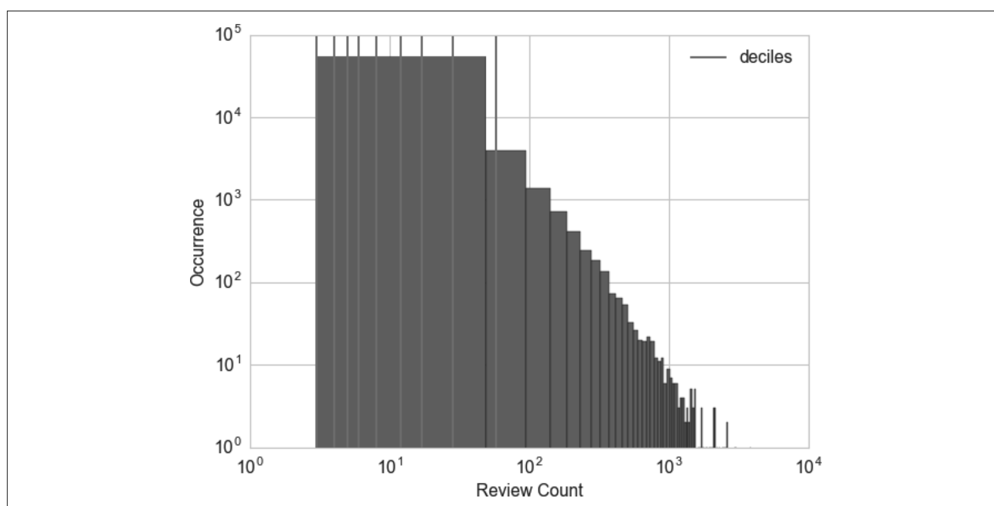


图 2-5：Yelp 点评数据集中的点评数量十分位数，*x* 轴和 *y* 轴都是对数标度

要计算分位数并将数据映射到分位数分箱中，可以使用 Pandas 库，如例 2-5 所示。`pandas.DataFrame.quantile` 和 `pandas.Series.quantile` 可以计算分位数。`pandas.qcut` 可以将数据映射为所需的分位数值。

例 2-5　通过分位数对计数值进行分箱

```
# 继续使用例2-3中的large_couts
>>> import pandas as pd

# 将计数值映射为分位数
```

```
>>> pd.qcut(large_counts, 4, labels=False)
array([1, 2, 3, 0, 0, 1, 1, 2, 2, 3, 3, 0, 0, 2, 1, 0, 3], dtype=int64)

# 计算实际的分位数值
>>> large_counts_series = pd.Series(large_counts)
>>> large_counts_series.quantile([0.25, 0.5, 0.75])
0.25     122.0
0.50     926.0
0.75    8286.0
dtype: float64
```

2.3　对数变换

上一节简要地介绍了通过取计数值的对数将数据映射到指数宽度分箱的方法。这一节，我们深入地研究一下这种方法。

对数函数是指数函数的反函数，它的定义是 $\log_a(a^x) = x$，其中 a 是个正的常数，x 可以是任意正数。因为 $a^0 = 1$，所以有 $\log_a(1) = 0$。这意味着对数函数可以将 (0, 1) 这个小区间中的数映射到 $(-\infty, 0)$ 这个包括全部负数的大区间上。函数 $\log_{10}(x)$ 可以将区间 [1, 10] 映射到 [0, 1]，将 [10, 100] 映射到 [1, 2]，以此类推。换言之，对数函数可以对大数值的范围进行压缩，对小数值的范围进行扩展。x 越大，$\log(x)$ 增长得越慢。

通过查看对数函数的图形（见图 2-6），可以更好地理解上面的内容。注意一下横轴上从 100 到 1000 的 x 值是如何被压缩到纵轴上从 2.0 到 3.0 的 y 值的，小于 100 的 x 值只占横轴的一小部分，但通过对数函数的映射，却占据了纵轴的剩余部分。

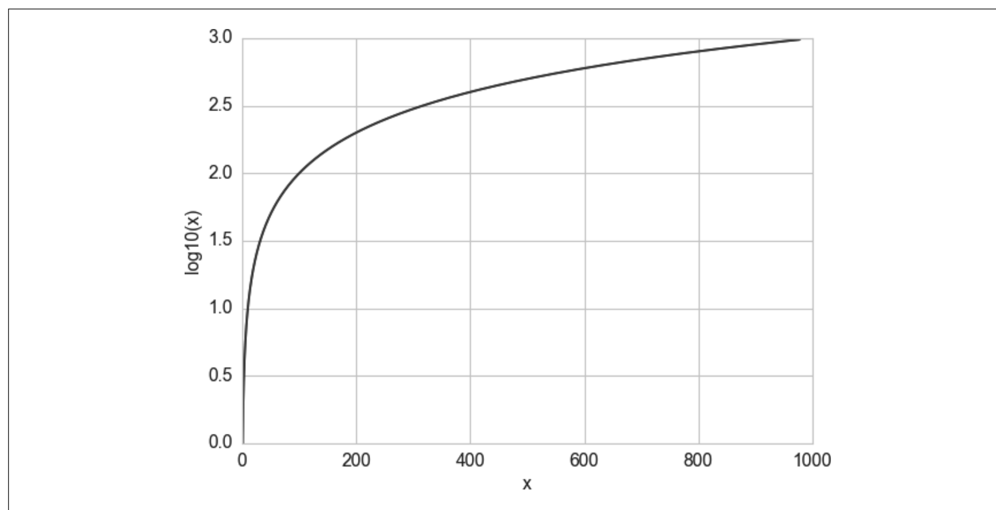


图 2-6：对数函数压缩高值区间并扩展低值区间

对于具有重尾分布的正数值的处理，对数变换是一个非常强大的工具。（与高斯分布相比，

重尾分布的概率质量更多地位于尾部。）它压缩了分布高端的长尾，使之成为较短的尾部，并将低端扩展为更长的头部。图 2-7 比较了对数变换前后的 Yelp 商家点评数量的直方图（见例 2-6），两幅图中的 y 轴都是正常（线性）标度。在下方的图形中，区间 (0.5, 1] 中的箱体间隔很大，是因为在 1 和 10 之间只有 10 个可能的整数计数值。请注意，初始的点评数量严重集中在低计数值区域，但有些异常值跑到了 4000 之外。经过对数变换之后，直方图在低计数值的集中趋势被减弱了，在 x 轴上的分布更均匀了一些。

例 2-6　对数变换前后的点评数量分布可视化

```
>>> fig, (ax1, ax2) = plt.subplots(2,1)
>>> biz_df['review_count'].hist(ax=ax1, bins=100)
>>> ax1.tick_params(labelsize=14)
>>> ax1.set_xlabel('review_count', fontsize=14)
>>> ax1.set_ylabel('Occurrence', fontsize=14)

>>> biz_df['log_review_count'].hist(ax=ax2, bins=100)
>>> ax2.tick_params(labelsize=14)
>>> ax2.set_xlabel('log10(review_count))', fontsize=14)
>>> ax2.set_ylabel('Occurrence', fontsize=14)
```

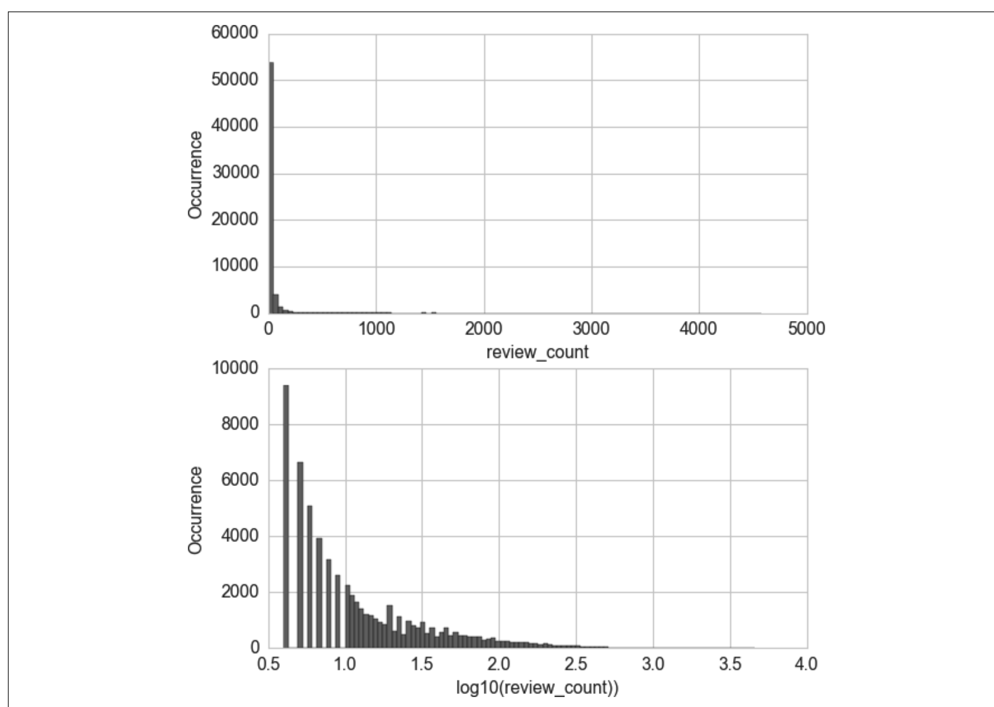


图 2-7：对数变换前（上）后（下）的 Yelp 商家点评数量比较

再来看一个例子——来自加州大学欧文分校机器学习库的在线新闻流行度数据集（Fernandes 等，2015）。

在线新闻流行度数据集的统计信息

- 数据集中包括了 Mashable 网站发表的 39 797 篇新闻文章，时间跨度为 2 年，共有 60 个特征。

我们的目标是使用这些特征来预测文章的流行度，流行度用社交媒体上的分享数表示。在这个例子中，我们只重点研究一个特征——文章中的单词个数。图 2-8 展示了这个特征在对数变换前后的直方图（见例 2-7）。请注意，在对数变换之后，特征分布更像是高斯分布了，但有一个例外，就是 0 长度文章（无内容）的数量有一个突变。

例 2-7　新闻文章流行度分布的可视化，使用对数变换和不使用对数变换

```
>>> fig, (ax1, ax2) = plt.subplots(2,1)
>>> df['n_tokens_content'].hist(ax=ax1, bins=100)
>>> ax1.tick_params(labelsize=14)
>>> ax1.set_xlabel('Number of Words in Article', fontsize=14)
>>> ax1.set_ylabel('Number of Articles', fontsize=14)

>>> df['log_n_tokens_content'].hist(ax=ax2, bins=100)
>>> ax2.tick_params(labelsize=14)
>>> ax2.set_xlabel('Log of Number of Words', fontsize=14)
>>> ax2.set_ylabel('Number of Articles', fontsize=14)
```

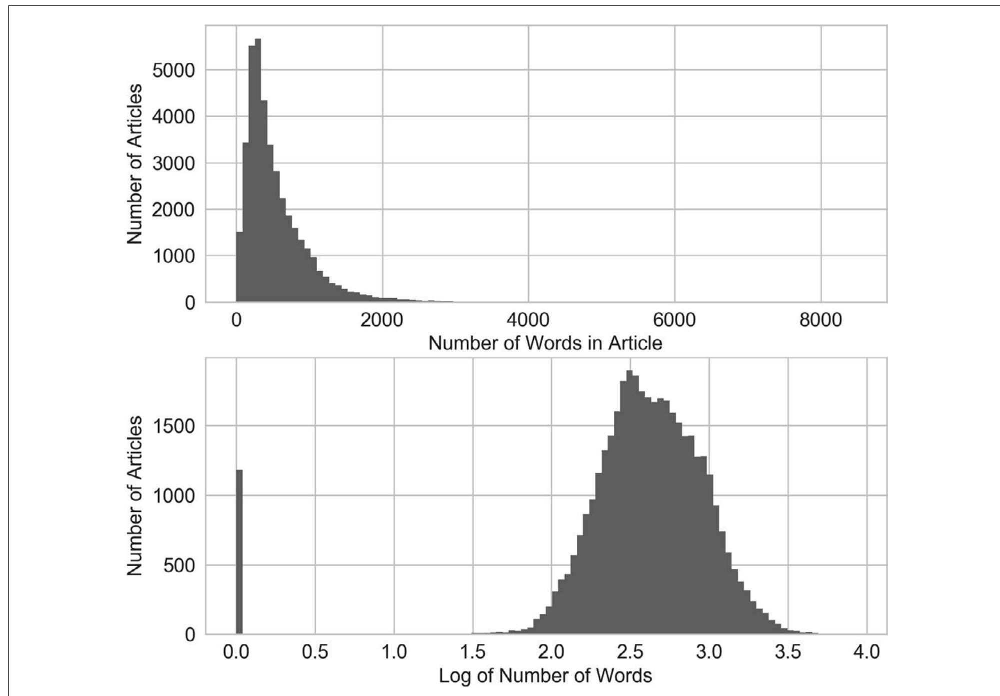


图 2-8：对数变换前（上）后（下）的 Mashable 新闻文章单词数比较

2.3.1 对数变换实战

我们看一下对数变换在监督式学习中有什么作用。本节会使用前面的两个数据集。对于 Yelp 点评数据集，我们使用点评数量来预测一个商家的平均评分（见例 2-8）。对于 Mashable 新闻文章数据集，我们使用文章中的单词数量来预测文章流行度。因为两个预测的输出都是连续型数值，所以我们可以使用简单线性回归来构造模型。我们使用 scikit-learn，分别在进行了对数变换和未进行对数变换的特征上进行 10-折交叉验证的线性回归。我们使用 R 方分数来评价模型，它衡量的是训练出的回归模型预测新数据的能力。良好的模型会有较高的 R 方分数。完美的模型能得到的最大 R 方分数是 1。R 方分数可以是负的，一个糟糕的模型可以得到任意低的负分。通过交叉验证，我们不仅能得到分数的估计值，还能得到它的方差，这有助于我们判断两种模型之间的差异是否是有意义的。

例 2-8　使用对数变换后的 Yelp 点评数量预测商家的平均评分

```
>>> import pandas as pd
>>> import numpy as np
>>> import json
>>> from sklearn import linear_model
>>> from sklearn.model_selection import cross_val_score

# 使用前面加载的Yelp点评数据框，计算Yelp点评数量的对数变换值。
# 注意，我们为原始点评数量加1，以免当点评数量为0时，对数运算结果得到负无穷大。
>>> biz_df['log_review_count'] = np.log10(biz_df['review_count'] + 1)

# 使用经过对数变换和未经对数变换的review_count特征，训练线性回归模型预测
# 一个商家的平均星级评分。比较两种模型的10-折交叉验证得分。
>>> m_orig = linear_model.LinearRegression()
>>> scores_orig = cross_val_score(m_orig, biz_df[['review_count']],
...                               biz_df['stars'], cv=10)
>>> m_log = linear_model.LinearRegression()
>>> scores_log = cross_val_score(m_log, biz_df[['log_review_count']],
...                              biz_df['stars'], cv=10)
>>> print("R-squared score without log transform: %0.5f (+/- %0.5f)"
...       % (scores_orig.mean(), scores_orig.std() * 2))
>>> print("R-squared score with log transform: %0.5f (+/- %0.5f)"
...       % (scores_log.mean(), scores_log.std() * 2))
R-squared score without log transform: -0.03683 (+/- 0.07280)
R-squared score with log transform: -0.03694 (+/- 0.07650)
```

从这个实验的结果可知，这两个简单模型（经过对数变换和未经对数变换）对目标变量的预测效果都非常差，经过对数变换的特征表现得要更糟糕一些。真让人失望！因为这两个模型都只使用了一个特征，所以表现不好也完全可以理解，但我们本来希望特征经过对数变换后会表现得好一些。

下面看看对数变换在在线新闻流行度数据集上所起的作用（见例 2-9）。

例 2-9　使用在线新闻流行度数据集中经对数变换后的单词个数预测文章流行度

```
# 从UCI下载在线新闻流行度数据集，然后使用Pandas将文件加载到数据框中。
>>> df = pd.read_csv('OnlineNewsPopularity.csv', delimiter=', ')

# 对n_tokens_content特征进行对数变换，这个特征表示的是一篇新闻文章中的单词
# （token）数量。
>>> df['log_n_tokens_content'] = np.log10(df['n_tokens_content'] + 1)

# 训练两个线性回归模型来预测文章的分享数，一个模型使用初始的特征，
# 另一个模型使用对数变换后的特征。
>>> m_orig = linear_model.LinearRegression()
>>> scores_orig = cross_val_score(m_orig, df[['n_tokens_content']],
...                               df['shares'], cv=10)
>>> m_log = linear_model.LinearRegression()
>>> scores_log = cross_val_score(m_log, df[['log_n_tokens_content']],
...                              df['shares'], cv=10)
>>> print("R-squared score without log transform: %0.5f (+/- %0.5f)"
...       % (scores_orig.mean(), scores_orig.std() * 2))
>>> print("R-squared score with log transform: %0.5f (+/- %0.5f)"
...       % (scores_log.mean(), scores_log.std() * 2))
R-squared score without log transform: -0.00242 (+/- 0.00509)
R-squared score with log transform: -0.00114 (+/- 0.00418)
```

置信区间还是重叠在一起，但使用对数变换后特征的模型表现得要比不用对数变换好一些。为什么在这个数据集上对数变换起到了积极作用？通过检查输入特征和目标值的散点图（见例 2-10），我们可以得到一些启发。从图 2-9 中下方的图可以看出，对数变换重组了 x 轴，对于那些目标变量值异常巨大（>200 000 个分享）的文章，对数变换将它们更多地拉向了 x 轴的右侧，这就为线性模型在输入特征空间的低值端争取了更多的“呼吸空间”。如果不进行对数变换（图 2-9 中上方的图），模型就会面临更大的压力，要在输入变化很小的情况下去拟合变化非常大的目标值。

例 2-10　新闻流行度预测问题中输入和输出相关性的可视化

```
>>> fig2, (ax1, ax2) = plt.subplots(2,1)
>>> ax1.scatter(df['n_tokens_content'], df['shares'])
>>> ax1.tick_params(labelsize=14)
>>> ax1.set_xlabel('Number of Words in Article', fontsize=14)
>>> ax1.set_ylabel('Number of Shares', fontsize=14)

>>> ax2.scatter(df['log_n_tokens_content'], df['shares'])
>>> ax2.tick_params(labelsize=14)
>>> ax2.set_xlabel('Log of the Number of Words in Article', fontsize=14)
>>> ax2.set_ylabel('Number of Shares', fontsize=14)
```

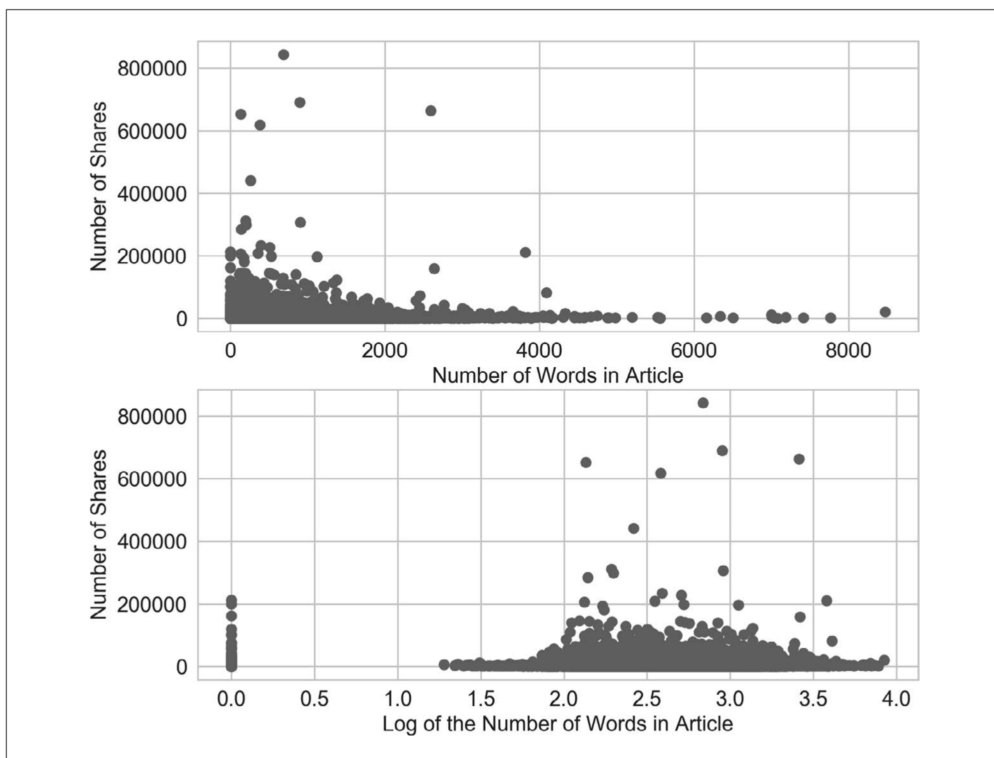


图 2-9：在线新闻流行度数据集中表示单词数（输入）和分享数（目标）关系的散点图。上图表示初始特征，下图是经过对数变换后的散点图

我们针对 Yelp 点评数据集也绘出散点图（见例 2-11），再和图 2-9 比较一下。图 2-10 看上去与图 2-9 非常不一样，平均星级评分是离散的，以半颗星的增量从 1 逐渐增加到 5。很高的评价数量（约 2500 个评价以上）确实与高的平均星级评分相关，但关系远非线性。没有明确的方法能够画出一条线来基于任意一个输入预测星级评分。基本上，根据图形我们就可以知道，评价数量和它的对数作为平均星级评分的线性预测器，表现得都非常糟糕。

例 2-11　Yelp 商家点评预测中的输入和输出相关性可视化

```
>>> fig, (ax1, ax2) = plt.subplots(2,1)
>>> ax1.scatter(biz_df['review_count'], biz_df['stars'])
>>> ax1.tick_params(labelsize=14)
>>> ax1.set_xlabel('Review Count', fontsize=14)
>>> ax1.set_ylabel('Average Star Rating', fontsize=14)

>>> ax2.scatter(biz_df['log_review_count'], biz_df['stars'])
>>> ax2.tick_params(labelsize=14)
>>> ax2.set_xlabel('Log of Review Count', fontsize=14)
>>> ax2.set_ylabel('Average Star Rating', fontsize=14)
```

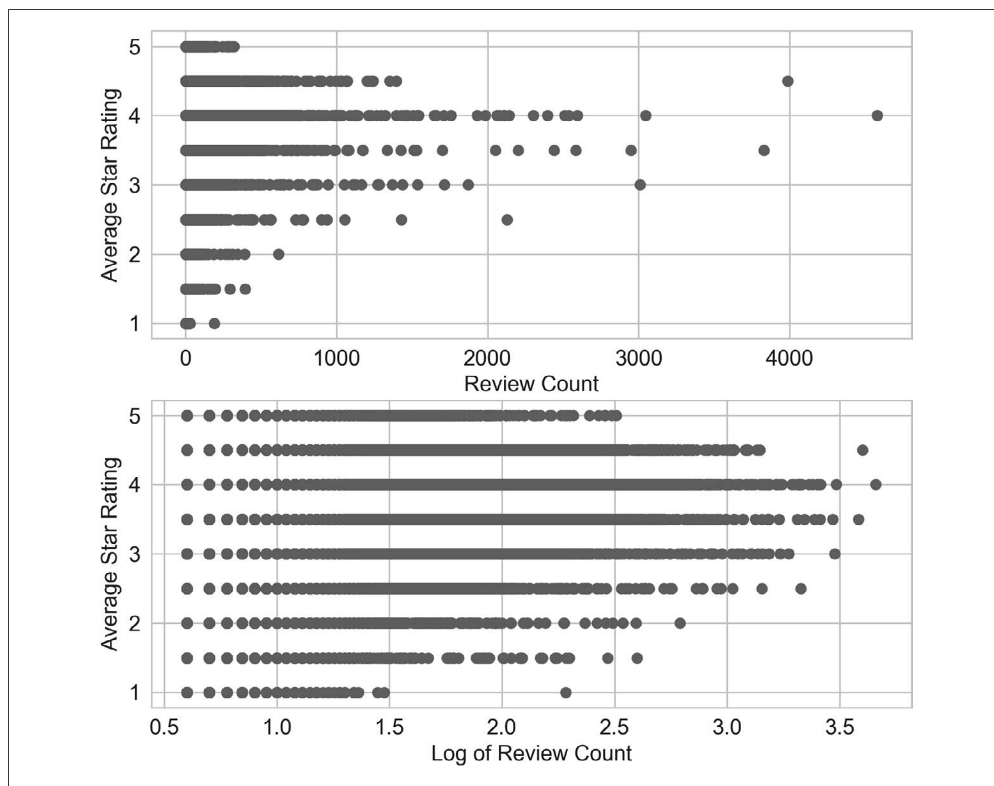
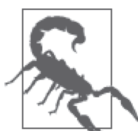


图 2-10：Yelp 点评数据集中表示点评数量（输入）和评价星级评分（目标）关系的散点图。上图使用原始点评数量，下图使用经对数变换后的点评数量



数据可视化的重要性

我们在两个不同的数据集中比较了对数变换的效果，并展示了数据可视化的重要性。这里，我们特意使用了非常简单的输入特征和目标变量，以便非常容易地对它们之间的关系进行可视化。像图 2-10 这样的图形可以立刻揭示出，我们选择的模型（线性模型）不可能表示相应的输入和目标之间的关系。另一方面，在给定平均星级评分的情况下，我们可以令人信服地做出点评数量的分布模型。在构建模型时，使用可视化方法查看一下输入和输出之间以及各个输入特征之间的关系是一种非常好的做法。

2.3.2 指数变换：对数变换的推广

指数变换是个变换族，对数变换只是它的一个特例。用统计学术语来说，它们都是**方差稳定化变换**。要理解为什么方差稳定是个好性质，可以考虑一下泊松分布。泊松分布是一种重尾分布，它的方差等于它的均值。因此，它的质心越大，方差就越大，重尾程度也越大。指数变换可以改变变量的分布，使得方差不再依赖于均值。例如，假设一个随机变量

X 具有泊松分布，如果通过取它的平方根对它进行变换，那么 $\tilde{X}=\sqrt{X}$ 的方差就近似是一个常数，而不是与均值相等。

图 2-11 表示出了 λ 对泊松分布的影响，λ 表示泊松分布的均值。当 λ 变大时，不仅整个分布模式向右移动，质量也更加分散，方差随之变大。

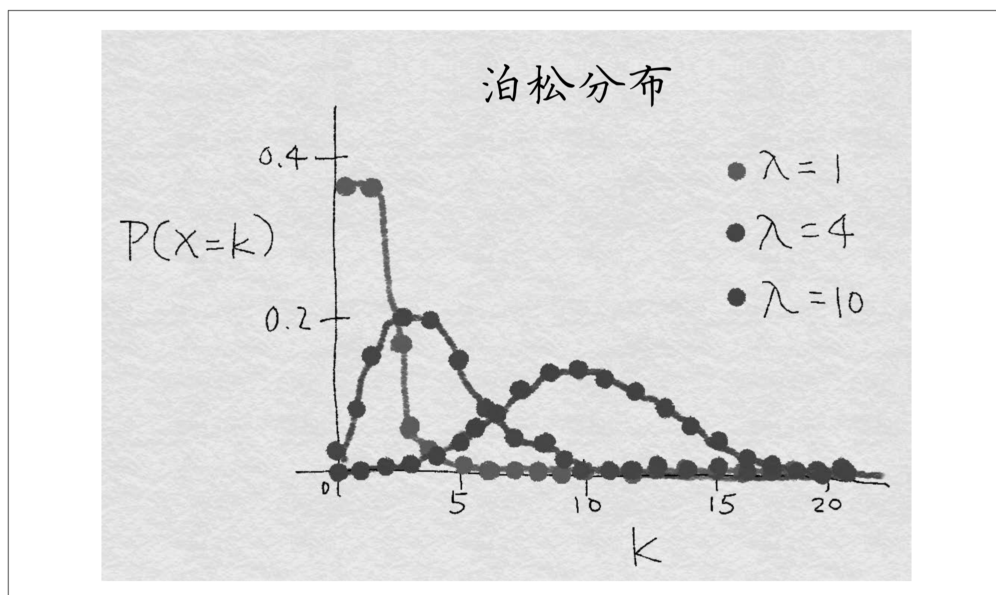


图 2-11：泊松分布的粗略表示，这是一个方差随均值变大的分布示例

平方根变换和对数变换都可以简单推广为 Box-Cox 变换：

$$\tilde{x}=\begin{cases}\dfrac{x^{\lambda}-1}{\lambda} & (\lambda\neq 0)\\ \ln(x) & (\lambda=0)\end{cases}$$

图 2-12 展示了 $\lambda=0$（对数变换）、$\lambda=0.25$、$\lambda=0.5$（平方根变换的一种缩放和平移形式）、$\lambda=0.75$ 和 $\lambda=1.5$ 时的 Box-Cox 变换。λ 小于 1 时，可以压缩高端值；λ 大于 1 时，起的作用是相反的。

只有当数据为正时，Box-Cox 公式才有效。对于非正数据，我们可以加上一个固定的常数，对数据进行平移。当应用 Box-Cox 变换或更广义的指数变换时，必须确定参数 λ 的值，这可以通过极大似然方法（找到能使变换后信号的高斯似然最大化的 λ 值）或贝叶斯方法来完成。Box-Cox 和广义指数变换的完整使用方法超出了本书的范围，感兴趣的读者可以在 Johnston 和 DiNardo 的《计量经济学方法》一书中获得更多的信息。幸运的是，SciPy 的 `stats` 包中有 Box-Cox 变换的实现方法，并包括找到最优变换参数的功能。例 2-12 演示了这种方法在 Yelp 点评数据集上的应用。

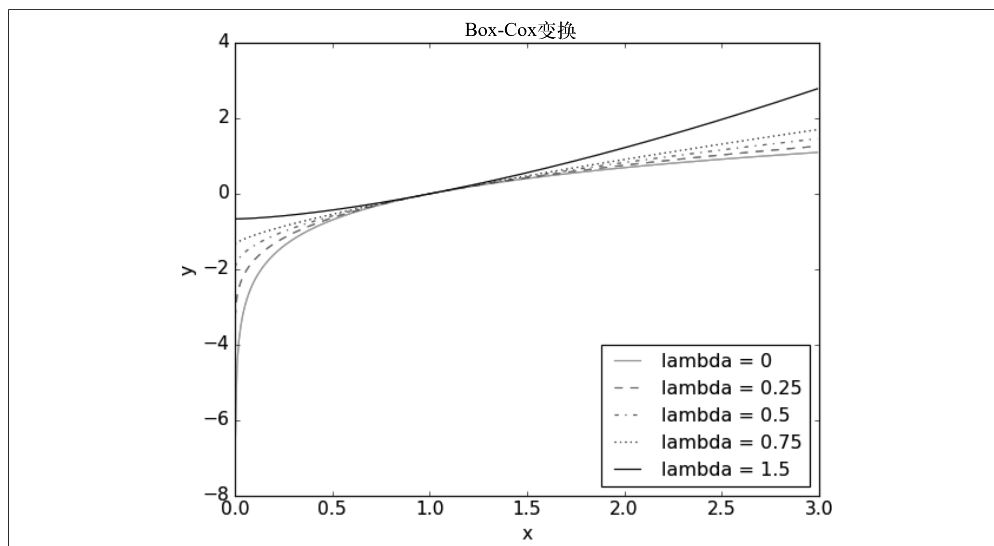


图 2-12：不同 λ 值的 Box-Cox 变换

例 2-12　对 Yelp 商家点评数量的 Box-Cox 变换

```
>>> from scipy import stats

# 接上一个例子，假设biz_df包含Yelp商家点评数据。
# Box-Cox变换假定输入数据都是正的。
# 检查数据的最小值以确定满足假定。
>>> biz_df['review_count'].min()
3

# 设置输入参数λ为0，使用对数变换（没有固定长度的位移）。
>>> rc_log = stats.boxcox(biz_df['review_count'], lmbda=0)
# 默认情况下，SciPy在实现Box-Cox转换时，会找出使得输出最接近于正态分布的λ参数。
>>> rc_bc, bc_params = stats.boxcox(biz_df['review_count'])
>>> bc_params
-0.4106510862321085
```

图 2-13 提供了初始点评数量和变换后点评数量在分布上的一个可视化比较（见例 2-13）。

例 2-13　初始、对数变换后和 Box-Cox 变换后的点评数量直方图可视化

```
>>> fig, (ax1, ax2, ax3) = plt.subplots(3,1)
# 初始点评数量直方图
>>> biz_df['review_count'].hist(ax=ax1, bins=100)
>>> ax1.set_yscale('log')
>>> ax1.tick_params(labelsize=14)
>>> ax1.set_title('Review Counts Histogram', fontsize=14)
>>> ax1.set_xlabel('')
>>> ax1.set_ylabel('Occurrence', fontsize=14)

# 对数变换后的点评数量
>>> biz_df['rc_log'].hist(ax=ax2, bins=100)
>>> ax2.set_yscale('log')
>>> ax2.tick_params(labelsize=14)
>>> ax2.set_title('Log Transformed Counts Histogram', fontsize=14)
```

```
>>> ax2.set_xlabel('')
>>> ax2.set_ylabel('Occurrence', fontsize=14)

# 最优Box-Cox变换后的点评数量
>>> biz_df['rc_bc'].hist(ax=ax3, bins=100)
>>> ax3.set_yscale('log')
>>> ax3.tick_params(labelsize=14)
>>> ax3.set_title('Box-Cox Transformed Counts Histogram', fontsize=14)
>>> ax3.set_xlabel('')
>>> ax3.set_ylabel('Occurrence', fontsize=14)
```

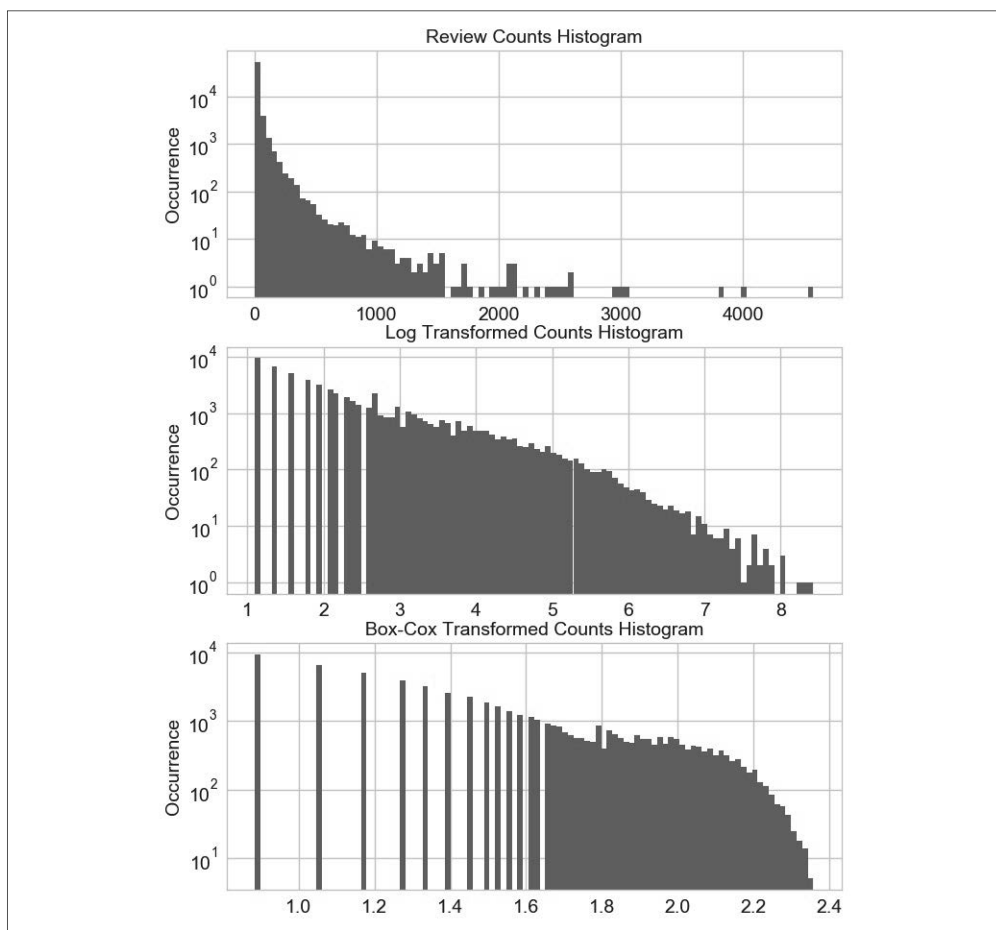


图 2-13：Box-Cox 变换后的 Yelp 商家点评数量直方图（下），以及初始点评数量直方图（上）和对数变换后的点评数量直方图（中）

概率图（probplot）是一种非常简单的可视化方法，用以比较数据的实际分布与理论分布，它本质上是一种表示实测分位数和理论分位数的关系的散点图。图 2-14 展示了 Yelp 点评数据的两种概率图，分别是初始点评数量、变换后点评数量的概率图，并和正态分布进行了对比（见例 2-14）。因为观测数据肯定是正的，而高斯分布可以是负的，所以在负数端，

实测分位数和理论分位数不可能匹配。因此，我们只关注正数部分。于是，我们可以看出与正态分布相比，初始的点评数量具有明显的重尾特征。（排序后的值可以达到 4000 以上，而理论分位数只能到达 4 左右。）普通对数变换和最优 Box-Cox 变换都可以将正尾部拉近正态分布。根据图形明显可以看出，相比对数变换，最优 Box-Cox 变换对尾部的压缩更强，它使得尾部变平，跑到了红色等值斜线下面。

例 2-14　初始和变换后点评数量的概率图，并和正态分布进行对比

```
>>> fig2, (ax1, ax2, ax3) = plt.subplots(3,1)
>>> prob1 = stats.probplot(biz_df['review_count'], dist=stats.norm, plot=ax1)
>>> ax1.set_xlabel('')
>>> ax1.set_title('Probplot against normal distribution')
>>> prob2 = stats.probplot(biz_df['rc_log'], dist=stats.norm, plot=ax2)
>>> ax2.set_xlabel('')
>>> ax2.set_title('Probplot after log transform')
>>> prob3 = stats.probplot(biz_df['rc_bc'], dist=stats.norm, plot=ax3)
>>> ax3.set_xlabel('Theoretical quantiles')
>>> ax3.set_title('Probplot after Box-Cox transform')
```

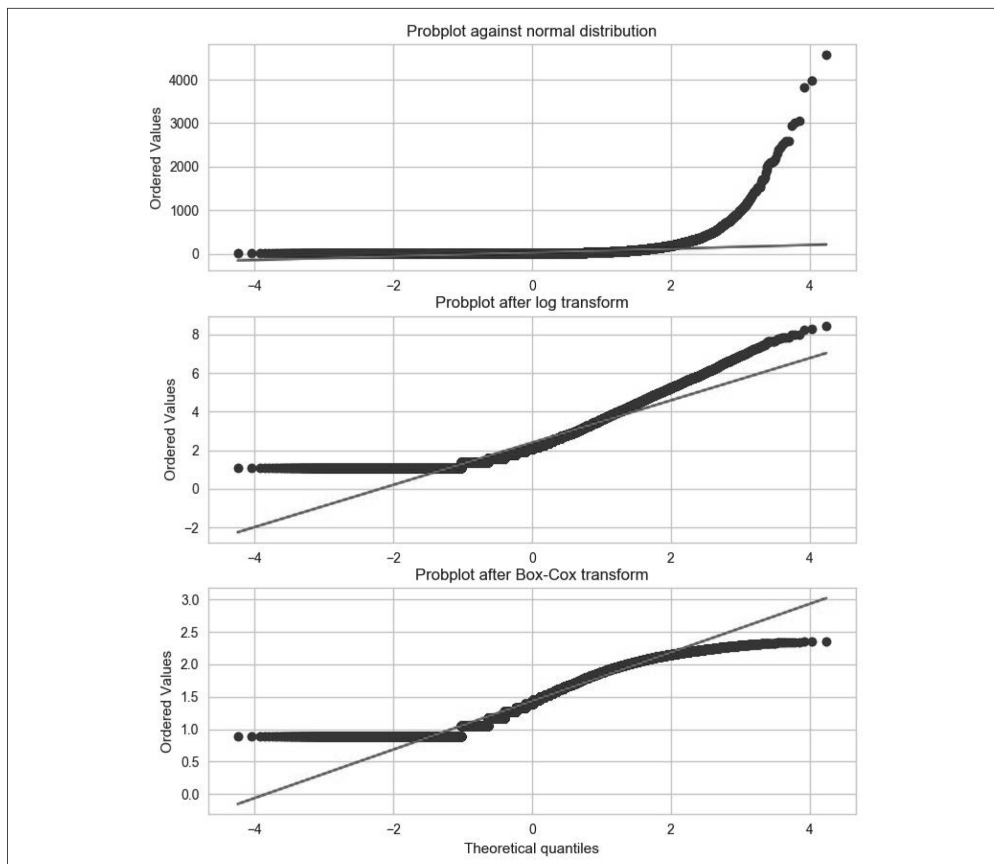


图 2-14：原始点评数量及变换后点评数量的分布与正态分布的对比

2.4 特征缩放/归一化

有些特征的值是有界限的，比如经度和纬度，但有些数值型特征可以无限制地增加，比如计数值。有些模型是输入的平滑函数，比如线性回归模型、逻辑回归模型或包含矩阵的模型，它们会受到输入尺度的影响。相反，那些基于树的模型则根本不在乎输入尺度有多大。如果模型对输入特征的尺度很敏感，就需要进行特征缩放。顾名思义，特征缩放会改变特征的尺度，有些人将其称为**特征归一化**。特征缩放通常对每个特征独立进行。下面讨论几种常用的特征缩放操作，每种操作都会产生一种不同的特征值分布。

2.4.1 min-max缩放

令 x 是一个独立的特征值（即某个数据点中的特征值），$\min(x)$ 和 $\max(x)$ 分别为这个特征在整个数据集中的最小值和最大值。min-max 缩放可以将所有特征值压缩（或扩展）到 [0, 1] 区间中。图 2-15 演示了这个过程。min-max 缩放的公式如下：

$$\frac{x \quad \min(x)}{\max(x) \quad \min(x)}$$

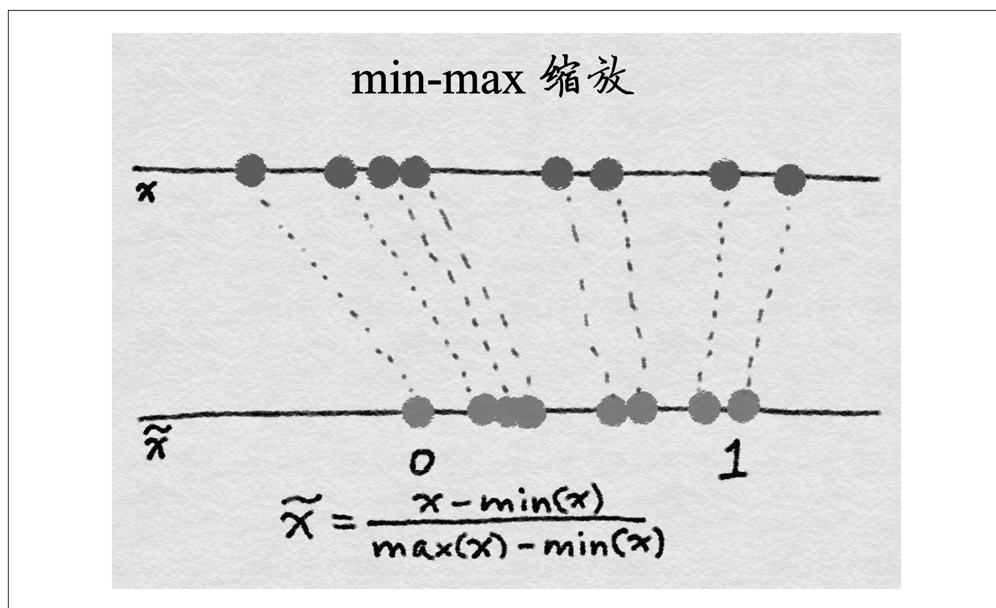


图 2-15：min-max 缩放示意图

2.4.2 特征标准化/方差缩放

特征标准化可以用下面的公式来定义：

$$\tilde{x} = \frac{x - \text{mean}(x)}{\text{sqrt}(\text{var}(x))}$$

它先减去特征的均值（对所有数据点），再除以方差，因此又称为**方差缩放**。缩放后的特征均值为 0，方差为 1。如果初始特征服从高斯分布，那么缩放后的特征也服从高斯分布。图 2-16 是这种标准化的示意图。

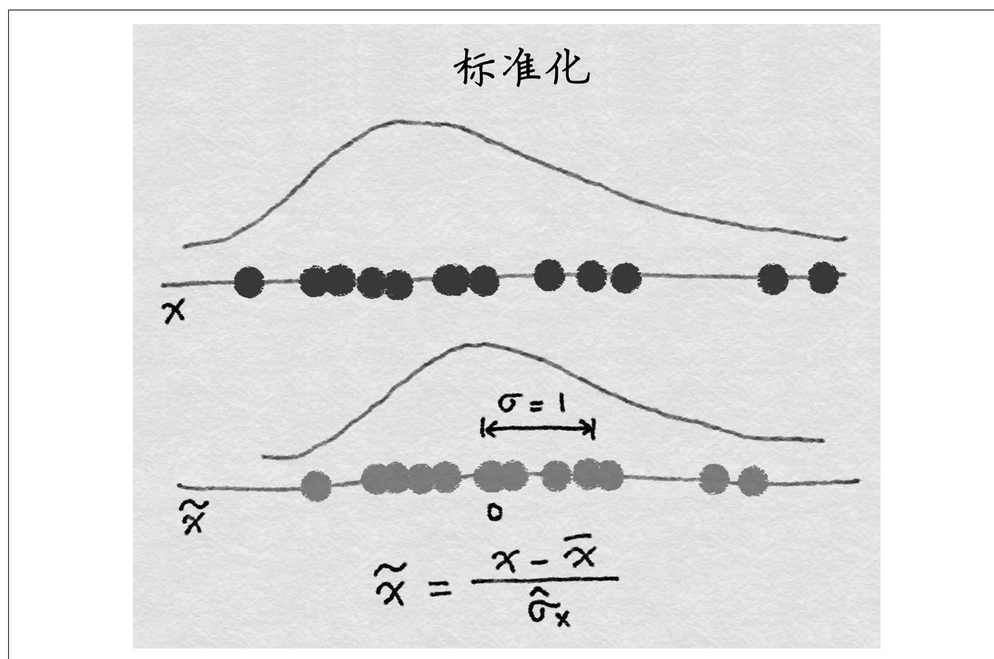
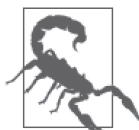


图 2-16：特征标准化示意图



不要“中心化”稀疏数据！

在稀疏特征上执行 min-max 缩放和标准化时一定要慎重，它们都会从原始特征值中减去一个量。对于 min-max 缩放，这个平移量是当前特征所有值中的最小值；对于标准化，这个量是均值。如果平移量不是 0，那么这两种变换会将一个多数元素为 0 的稀疏特征向量变成密集特征向量。根据实现方式的不同，这种改变会给分类器带来巨大的计算负担（按照现在的表示方法，特征向量中包含没有出现在一篇文档中的所有单词，不用说，这种特征向量变为密集向量是非常可怕的）。词袋就是一种稀疏的表示方式，大多数分类算法的实现都针对稀疏输入进行了优化。

2.4.3 ℓ^2归一化

这种归一化技术是将初始特征值除以一个称为 ℓ^2 范数的量，ℓ^2 范数又称为欧几里得范数，

它的定义如下：

$$\tilde{x} = \frac{x}{\|x\|_2}$$

ℓ^2 范数是坐标空间中向量长度的一种测量。它的定义可以根据著名的毕达哥拉斯定理（给定一个直角三角形两条直角边的长度，可以求出斜边的长度）导出：

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_m^2}$$

ℓ^2 范数先对所有数据点中该特征的值的平方求和，然后算出平方根。经过 ℓ^2 归一化后，特征列的范数就是 1。有时候这种处理又称为 ℓ^2 缩放。（不严格地说，**缩放**意味着乘以一个常数，而**归一化**可以包括多种操作。）图 2-17 演示了 ℓ^2 归一化的过程。

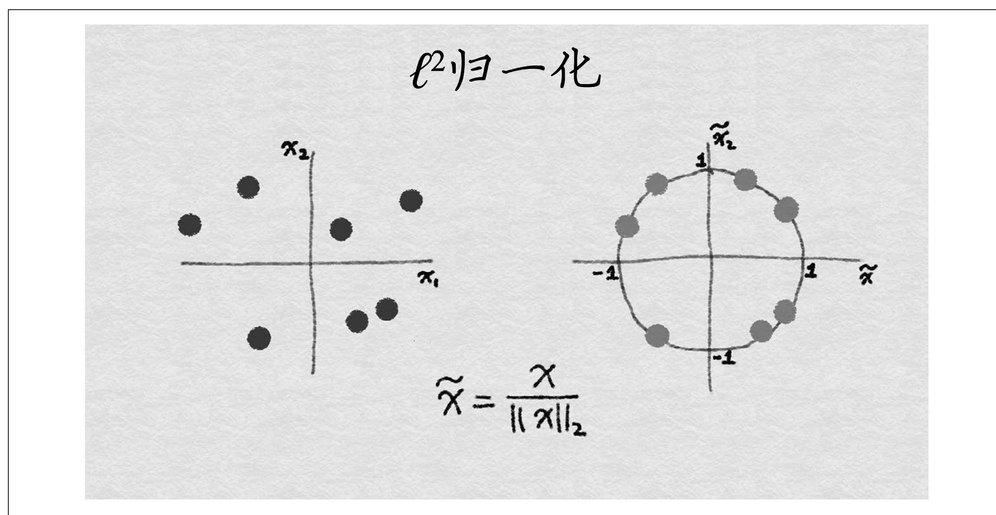


图 2-17：ℓ^2 特征归一化示意图



数据空间与特征空间

请注意，图 2-17 的演示中使用的是数据空间，不是特征空间。除了对特征进行 ℓ^2 归一化，也可以对数据点进行 ℓ^2 归一化，这会得到带有单位范数（范数为 1）的数据向量。参见 3.1.1 节中关于数据向量和特征向量互补关系的讨论。

不论使用何种缩放方法，特征缩放总是将特征除以一个常数（称为**归一化常数**）。因此，它不会改变单特征分布的形状。我们使用在线新闻文章单词数量来说明这一点（见例 2-15）。

例 2-15　特征缩放示例

```
>>> import pandas as pd
>>> import sklearn.preprocessing as preproc
```

```
# 加载在线新闻流行度数据集
>>> df = pd.read_csv('OnlineNewsPopularity.csv', delimiter=', ')

# 查看原始数据——文章中的单词数量
>>> df['n_tokens_content'].as_matrix()
array([ 219.,  255.,  211., ...,  442.,  682.,  157.])

# min-max缩放
>>> df['minmax'] = preproc.minmax_scale(df[['n_tokens_content']])
>>> df['minmax'].as_matrix()
array([ 0.02584376,  0.03009205,  0.02489969, ...,  0.05215955,
        0.08048147,  0.01852726])

# 标准化——注意根据标准化的定义，有些结果会是负的
>>> df['standardized'] = preproc.StandardScaler().fit_transform(df[['n_tokens_
content']])
>>> df['standardized'].as_matrix()
array([-0.69521045, -0.61879381, -0.71219192, ..., -0.2218518 ,
        0.28759248, -0.82681689])

# L2-归一化
>>> df['l2_normalized'] = preproc.normalize(df[['n_tokens_content']], axis=0)
>>> df['l2_normalized'].as_matrix()
array([ 0.00152439,  0.00177498,  0.00146871, ...,  0.00307663,
        0.0047472 ,  0.00109283])
```

我们还可以对通过其他特征缩放方法得到的数据分布进行可视化（见图 2-18）。如例 2-16 所示，与对数变换不同，特征缩放不改变分布的形状，只有数据尺度发生了变化。

例 2-16　绘制原始数据和缩放后数据的直方图

```
>>> fig, (ax1, ax2, ax3, ax4) = plt.subplots(4,1)
>>> fig.tight_layout()
>>> df['n_tokens_content'].hist(ax=ax1, bins=100)
>>> ax1.tick_params(labelsize=14)
>>> ax1.set_xlabel('Article word count', fontsize=14)
>>> ax1.set_ylabel('Number of articles', fontsize=14)

>>> df['minmax'].hist(ax=ax2, bins=100)
>>> ax2.tick_params(labelsize=14)
>>> ax2.set_xlabel('Min-max scaled word count', fontsize=14)
>>> ax2.set_ylabel('Number of articles', fontsize=14)

>>> df['standardized'].hist(ax=ax3, bins=100)
>>> ax3.tick_params(labelsize=14)
>>> ax3.set_xlabel('Standardized word count', fontsize=14)
>>> ax3.set_ylabel('Number of articles', fontsize=14)

>>> df['l2_normalized'].hist(ax=ax4, bins=100)
>>> ax4.tick_params(labelsize=14)
>>> ax4.set_xlabel('L2-normalized word count', fontsize=14)
>>> ax4.set_ylabel('Number of articles', fontsize=14)
```

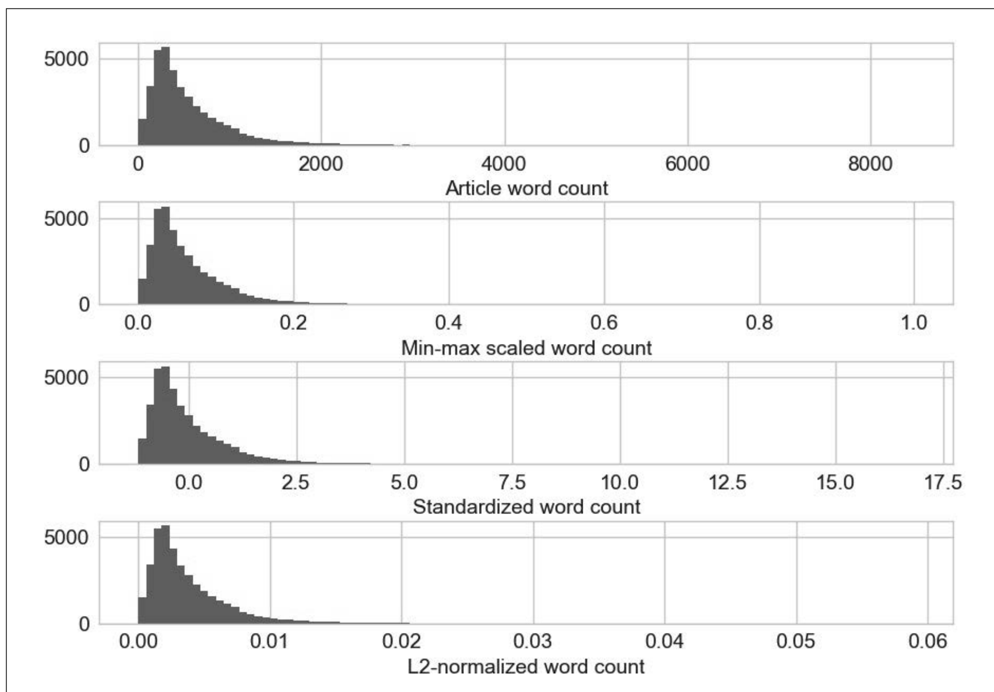


图 2-18：原始及缩放后的新闻文章单词数量。注意只有 x 轴的尺度发生了变化，特征缩放后的分布形状保持不变

当一组输入特征的尺度相差很大时，就需要进行特征缩放。例如，一个人气很高的商业网站的日访问量可能是几十万次，而实际购买行为可能只有几千次。如果这两个特征都被模型所使用，那么模型就需要在确定如何使用它们时先平衡一下尺度。如果输入特征的尺度差别非常大，就会对模型训练算法带来数值稳定性方面的问题。在这种情况下，就应该对特征进行标准化。第 4 章将详细介绍特征缩放在自然文本处理中的应用，并给出几个使用示例。

2.5　交互特征

两个特征的乘积可以组成一对简单的**交互特征**，这种相乘关系可以用逻辑操作符 AND 来类比，它可以表示出由一对条件形成的结果："该购买行为来自于邮政编码为 98121 的地区" AND "用户年龄在 18 和 35 岁之间"。这种特征在基于决策树的模型中极其常见，在广义线性模型中也经常使用。

简单线性模型使用独立输入特征 $x_1, x_2, \cdots, x_n$ 的线性组合来预测结果变量 y：

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

很容易对线性模型进行扩展，使之包含输入特征的两两组合，如下所示：

$$y = w_1x_1 + w_2x_2 + \cdots + w_nx_n + w_{1,1}x_1x_1 + w_{1,2}x_1x_2 + w_{1,3}x_1x_3 + \cdots$$

这样，就可以捕获特征之间的交互作用，因此这些特征对就称为**交互特征**。如果 x_1 和 x_2 是二值特征，那么它们的积 x_1x_2 就是逻辑函数 x_1 AND x_2。如果我们的问题是基于客户档案信息来预测客户偏好，那么在这个例子中，除了根据用户的年龄或地点这些单独特征来进行预测，还可以使用交互特征来根据用户位于某个年龄段并位于某个特定地点来进行预测。

在例 2-17 中，我们使用了来自 UCI 在线新闻流行度数据集中的交互特征对来预测每篇新闻文章的分享数量。从结果可以看出，与单一特征相比，交互特征使准确率有了一定提升。两种方法的表现都优于例 2-9，其中我们使用（未经对数变换和对数变换后的）文章中的单词数作为唯一预测特征。

例 2-17　预测中的交互特征示例

```
>>> from sklearn import linear_model
>>> from sklearn.model_selection import train_test_split
>>> import sklearn.preprocessing as preproc

# 假设df是一个Pandas数据框，其中包含了UCI在线新闻流行度数据集
>>> df.columns
Index(['url', 'timedelta', 'n_tokens_title', 'n_tokens_content',
       'n_unique_tokens', 'n_non_stop_words', 'n_non_stop_unique_tokens',
       'num_hrefs', 'num_self_hrefs', 'num_imgs', 'num_videos',
       'average_token_length', 'num_keywords', 'data_channel_is_lifestyle',
       'data_channel_is_entertainment', 'data_channel_is_bus',
       'data_channel_is_socmed', 'data_channel_is_tech',
       'data_channel_is_world', 'kw_min_min', 'kw_max_min', 'kw_avg_min',
       'kw_min_max', 'kw_max_max', 'kw_avg_max', 'kw_min_avg', 'kw_max_avg',
       'kw_avg_avg', 'self_reference_min_shares', 'self_reference_max_shares',
       'self_reference_avg_sharess', 'weekday_is_monday', 'weekday_is_tuesday',
       'weekday_is_wednesday', 'weekday_is_thursday', 'weekday_is_friday',
       'weekday_is_saturday', 'weekday_is_sunday', 'is_weekend', 'LDA_00',
       'LDA_01', 'LDA_02', 'LDA_03', 'LDA_04', 'global_subjectivity',
       'global_sentiment_polarity', 'global_rate_positive_words',
       'global_rate_negative_words', 'rate_positive_words',
       'rate_negative_words', 'avg_positive_polarity', 'min_positive_polarity',
       'max_positive_polarity', 'avg_negative_polarity',
       'min_negative_polarity', 'max_negative_polarity', 'title_subjectivity',
       'title_sentiment_polarity', 'abs_title_subjectivity',
       'abs_title_sentiment_polarity', 'shares'],
      dtype='object')

# 选择与内容有关的特征作为模型的单一特征，忽略那些衍生特征
>>> features = ['n_tokens_title', 'n_tokens_content',
...             'n_unique_tokens', 'n_non_stop_words', 'n_non_stop_unique_tokens',
...             'num_hrefs', 'num_self_hrefs', 'num_imgs', 'num_videos',
...             'average_token_length', 'num_keywords', 'data_channel_is_lifestyle',
```

```
...             'data_channel_is_entertainment', 'data_channel_is_bus',
...             'data_channel_is_socmed', 'data_channel_is_tech',
...             'data_channel_is_world']

>>> X = df[features]
>>> y = df[['shares']]

# 创建交互特征对，跳过固定偏移项
>>> X2 = preproc.PolynomialFeatures(include_bias=False).fit_transform(X)
>>> X2.shape
(39644, 170)

# 为两个特征集创建训练集和测试集
>>> X1_train, X1_test, X2_train, X2_test, y_train, y_test = \
...     train_test_split(X, X2, y, test_size=0.3, random_state=123)

>>> def evaluate_feature(X_train, X_test, y_train, y_test):
...     """Fit a linear regression model on the training set and
...     score on the test set"""
...     model = linear_model.LinearRegression().fit(X_train, y_train)
...     r_score = model.score(X_test, y_test)
...     return (model, r_score)

# 在两个特征集上训练模型并比较R方分数
>>> (m1, r1) = evaluate_feature(X1_train, X1_test, y_train, y_test)
>>> (m2, r2) = evaluate_feature(X2_train, X2_test, y_train, y_test)
>>> print("R-squared score with singleton features: %0.5f" % r1)
>>> print("R-squared score with pairwise features: %0.10f" % r2)
R-squared score with singleton features: 0.00924
R-squared score with pairwise features: 0.0113276523
```

交互特征的构造非常简单，使用起来却代价不菲。如果线性模型中包含有交互特征对，那它的训练时间和评分时间就会从 $O(n)$ 增加到 $O(n^2)$，其中 n 是单一特征的数量。

有若干种方法可以绕过高阶交互特征所带来的计算成本。我们可以在构造出所有交互特征之后再执行特征选择，或者，也可以更加精心地设计出少量复杂特征。

这两种策略各有千秋。特征选择使用计算手段为一个具体问题选择出最佳特征。（这种技术并不局限于交互特征。）但是，一些特征选择技术仍然需要使用大量特征去训练多个模型。

精心设计的复杂特征需要昂贵的成本，所以数量不能太多，它们可以减少模型的训练时间，但特征本身会消耗很多计算能力，这增加了模型评分阶段的计算成本。第 8 章中会给出几个人工设计（或机器学习）的复杂特征的优秀示例。下面介绍几种特征选择技术。

2.6 特征选择

特征选择技术可以精简掉无用的特征，以降低最终模型的复杂性，它的最终目的是得到一个简约模型，在不降低预测准确率或对预测准确率影响不大的情况下提高计算速度。为了得到

这样的模型，有些特征选择技术需要训练不止一个待选模型。换言之，特征选择不是为了减少训练时间（实际上，一些技术会增加总体训练时间），而是为了减少模型评分时间。

粗略地说，特征选择技术可以分为以下三类。

过滤

过滤技术对特征进行预处理，以除去那些不太可能对模型有用处的特征。例如，我们可以计算出每个特征与响应变量之间的相关性或互信息，然后过滤掉那些在某个阈值之下的特征。第 3 章将讨论用于文本特征的这种技术。过滤技术的成本比下面描述的打包技术低廉得多，但它们没有考虑我们要使用的模型，因此，它们有可能无法为模型选择出正确的特征。我们最好谨慎地使用预过滤技术，以免在有用特征进入到模型训练阶段之前不经意地将其删除。

打包方法

这些技术的成本非常高昂，但它们可以试验特征的各个子集，这意味着我们不会意外地删除那些本身不提供什么信息但和其他特征组合起来却非常有用的特征。打包方法将模型视为一个能对推荐的特征子集给出合理评分的黑盒子。它们使用另外一种方法迭代地对特征子集进行优化。

嵌入式方法

这种方法将特征选择作为模型训练过程的一部分。例如，特征选择是决策树与生俱来的一种功能，因为它在每个训练阶段都要选择一个特征来对树进行分割。另一个例子是 ℓ^1 正则项，它可以添加到任意线性模型的训练目标中。ℓ^1 正则项鼓励模型使用更少的特征，而不是更多的特征，所以又称为模型的稀疏性约束。嵌入式方法将特征选择整合为模型训练过程的一部分。它们不如打包方法强大，但成本也远不如打包方法那么高。与过滤技术相比，嵌入式方法可以选择出特别适合某种模型的特征。从这个意义上说，嵌入式方法在计算成本和结果质量之间实现了某种平衡。

特征选择的完整操作方法超出了本书范围，对此感兴趣的读者可以参考 Guyon 和 Elisseeff 在 2003 年做的文献综述。

2.7 小结

本章讨论了几种常用的数值型特征工程技术，比如区间量化、缩放（即归一化）、对数变换（指数变换的一种）和交互特征，并对特征选择技术进行了简介，这对于处理大量交互特征是必需的。在统计机器学习中，所有数据最终都会转化为数值型特征。因此，所有特征工程最终都会归结为某种数值型特征工程技术。请熟练掌握本章介绍的这些技术，最终完成特征工程吧!

2.8 参考文献

Bertin-Mahieux, Thierry, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset [C]. Proceedings of the 12th International Society for Music Information Retrieval Conference (2011): 591–596.

Fernandes, K., P. Vinagre, and P. Cortez. A Proactive Intelligent Decision Support System for Predicting the Popularity of Online News [C]. Proceedings of the 17th Portuguese Conference on Artificial Intelligence (2015): 535–546.

Guyon, Isabell, and André Elisseeff. An Introduction to Variable and Feature Selection [J]. Journal of Machine Learning Research Special Issue on Variable and Feature Selection 3 (2003): 1157–1182.

Johnston, Jack, and John DiNardo. Econometric Methods [M]. 4th ed. New York: McGraw Hill, 1997.

第3章

文本数据：扁平化、过滤和分块

如果你想设计一种算法来分析以下一段文本，应该怎么做呢？

> Emma knocked on the door. No answer. She knocked again and waited. There was a large maple tree next to the house. Emma looked up the tree and saw a giant raven perched at the treetop. Under the afternoon sun, the raven gleamed magnificently. Its beak was hard and pointed, its claws sharp and strong. It looked regal and imposing. It reigned the tree it stood on. The raven was looking straight at Emma with its beady black eyes. Emma felt slightly intimidated. She took a step back from the door and tentatively said, “Hello?”

这段文本包含了很多信息。我们可以知道，其中有个叫 Emma 的人，还有一只乌鸦、一栋房子和一棵树。Emma 试图进入这栋房子，却看到了乌鸦。乌鸦很威武雄壮，并注意到了 Emma，Emma 有点胆怯，但仍鼓起勇气打了个招呼。

那么，从这个信息宝藏中我们能提取出哪些显著特征呢？首先，我们似乎应该提取出主要角色的名称，Emma 和乌鸦。其次，我们也不能忽略房屋、门和树这种环境因素。但是，那些关于乌鸦的描写应该怎么办呢？ Emma 的那些行为——敲门、后退和打招呼——该怎么处理呢？

本章将介绍用于文本的特征工程的基础知识。我们从**词袋**开始，这是基于单词数量统计的最简单的文本特征表示方法。与词袋密切相关的一种转换技术是 **tf-idf**，它本质上是一种特征缩放技术，我们将用专门的一章（下一章）来充分讨论这种技术。在本章中，我们首先介绍如何从文本中提取特征，然后再研究一下如何过滤和清洁这些特征。

3.1 元素袋：将自然文本转换为扁平向量

不管是建立机器学习模型，还是构建特征，既简单又可以解释的结果自然是非常好的。简单的事情很容易尝试，相对于复杂的特征和模型，可解释的特征和模型更易于调试。虽然简单明了的特征不一定会得到最准确的模型，但从简单开始并且仅在绝对必要的时候才添加复杂性总是没错的。

对于文本数据，我们可以从一个单词数量的统计列表开始，这称为词袋（bag-of-words，BoW）。这个单词数量列表并不试图找出有意义的实体，比如 Emma 或 raven（乌鸦）。但这两个单词在我们的示例文本中被提及了多次，它们出现的次数远高于那些随机出现的单词，比如“hello”。对于像文档分类这样的简单任务来说，单词数量统计通常就够用了。这种技术还可以用于信息提取，它的目标是提取出一组与查询文本相关的文档。这两种任务都可以凭借单词级别的特征圆满地完成，因为特定词是否存在于文档中这个指标可以很好地标识文档的主题内容。

3.1.1 词袋

在词袋特征化中，一篇文本文档被转化为一个计数向量。（向量就是 n 个数值的集合。）这个计数向量包含词汇表中所有可能出现的单词。如果某个单词（比如“aardvark”）在文档中出现了 3 次，那么特征向量在对应于这个单词的位置就有一个计数值 3。如果词汇表中的某个单词没有出现在文档中，那么它的计数值就是 0。例如，文本“it is a puppy and it is extremely cute”具有图 3-1 中的词袋表示。

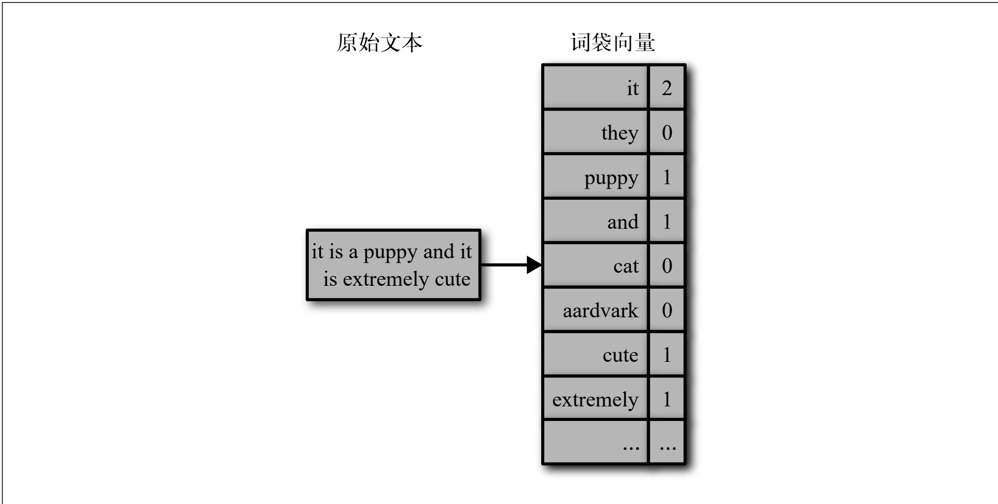


图 3-1：将原始文本转换为词袋表示

词袋将一个文本文档转换为一个扁平向量。之所以说这个向量是“扁平”的，是因为它

不包含原始文本中的任何结构。原始文本是一个单词序列，但词袋中没有任何序列，它只记录每个单词在文本中出现的次数。因此，如图 3-2 所示，向量中单词的顺序根本不重要，只要它在数据集的所有文档之间保持一致即可。词袋也不表示任何单词层次。例如，“animal”这个概念包括“dog”“cat”“raven”等，但在词袋表示中，这些单词在向量中都是平等的元素。

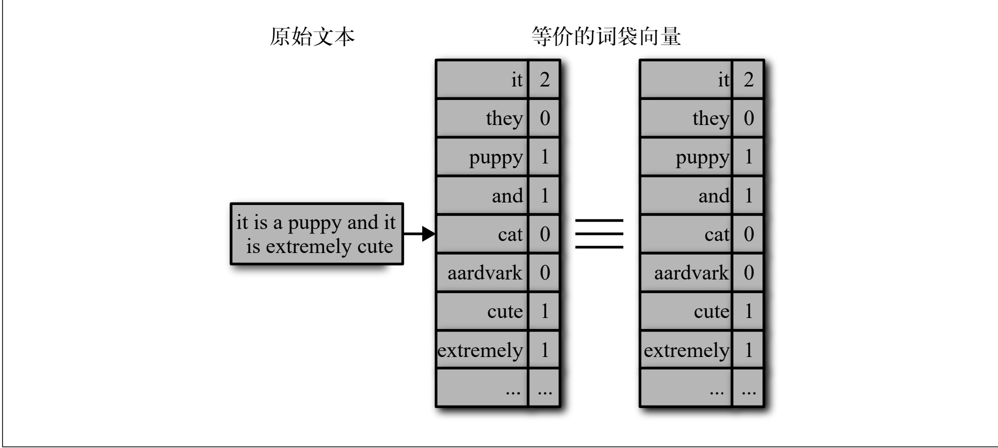


图 3-2：两个等价的词袋向量

在词袋表示中，重要的是特征空间中的数据分布。在词袋向量中，每个单词都是向量的一个维度。如果词汇表中有 *n* 个单词，那么一篇文档就是 *n* 维空间中的一个点[1]。对于超过二维或三维的任何分布，很难将其可视化，所以只好驰骋我们的想象了。图 3-3 展示了由单词“puppy”和“cute”所构成的二维特征空间，以及我们的例句在这个空间中的样子。

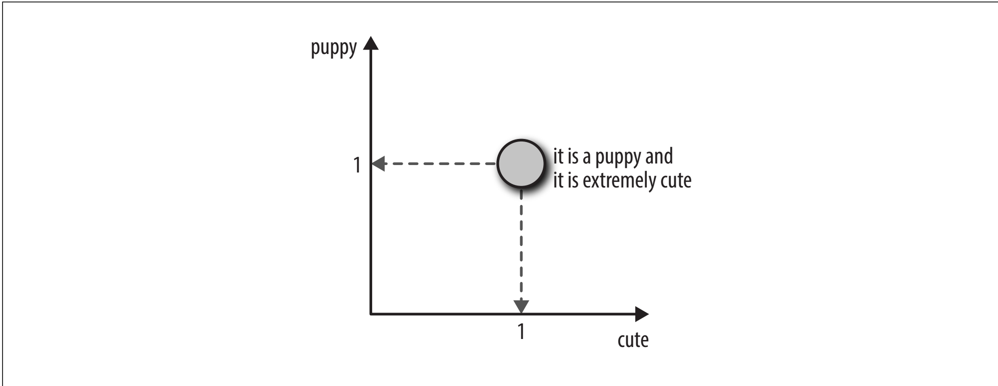


图 3-3：二维特征空间中的示例文本文档示意图

注 1：有时人们会使用“文档向量”这个术语。向量从原点出发，终于一个特定的点。对我们来说，“向量”和“点”是一回事。

图 3-4 展示了三维空间中的 3 个句子，三维空间由单词“puppy”“extremely”和“cute”构成。

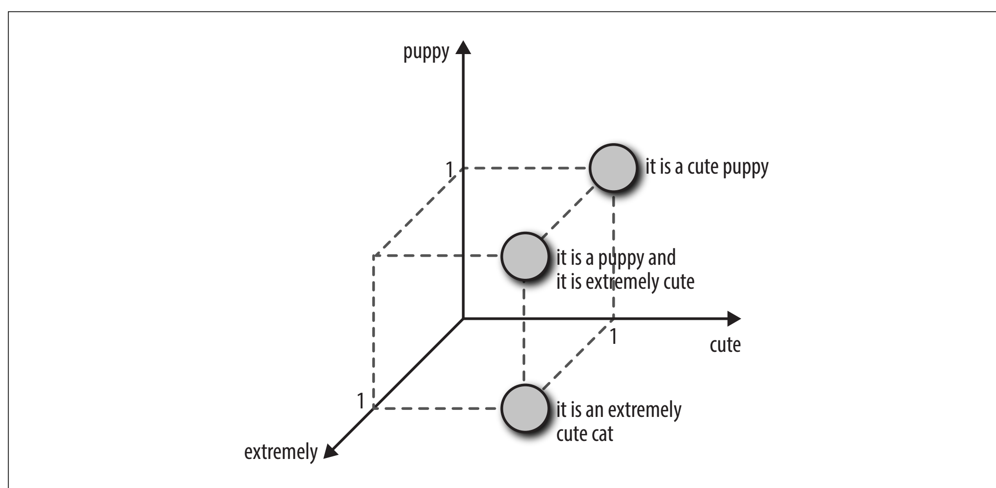


图 3-4：三维特征空间中的 3 个句子

这两幅图表示的都是特征空间中的数据向量。坐标轴表示单词，也就是词袋表示中的特征，空间中的点表示数据点（文本文档）。有时候，在**数据**空间中查看**特征**向量也能获取很多信息。在数据空间中，特征向量中包含的是每个数据点中各个特征的值。坐标轴表示各个数据点，其中的点表示特征向量。图 3-5 给出了一个这样的例子。通过对文本文档的词袋特征化，一个特征就是一个单词，一个特征向量由这个单词在每篇文档中出现的次数组成。这样，一个单词就可以表示为“文档袋”。正如我们将在第 4 章中看到的，这些文档袋向量来自于词袋向量的矩阵转置。

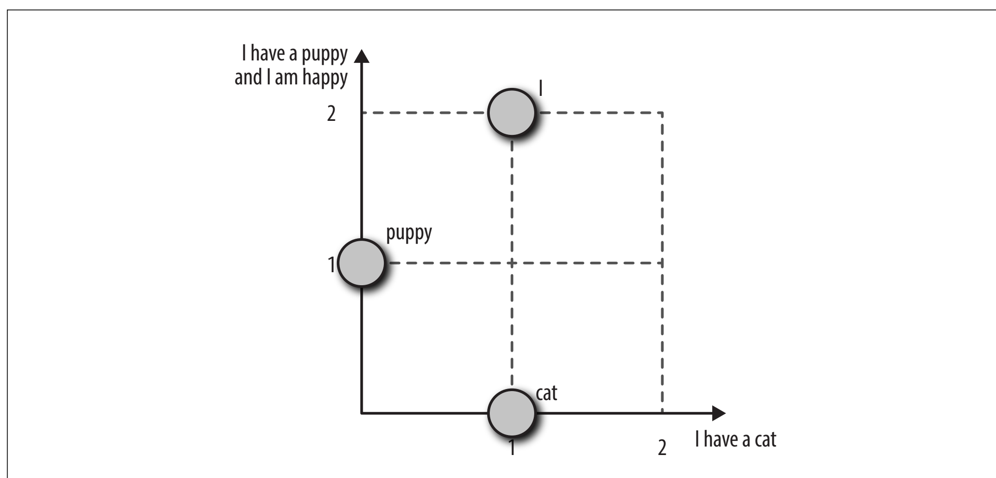


图 3-5：文档空间中的词向量

词袋并非完美无缺，将句子分解为单词会破坏语义。例如，“not bad”在语义上是“decent”，甚至是“good”（特别是在英式英语里）。但“not”和“bad”被分开后表示的是一种否定和负面的情感。“toy dog”和“dog toy”是差别很大的两种东西（除非是玩具狗的狗玩具），但拆成“toy”和“dog”这两个单词后，都失去了原来的意义。我们还可以轻松地举出很多其他的例子。我们随后要介绍的 *n* 元词袋可以在某种程度上解决这种问题，但不是根本的解决方案。我们应该知道，词袋是一种简单而有效的启发式方法，但离正确的文本语义理解还相去甚远。

3.1.2　*n*元词袋

n 元词袋（bag-of-n-grams）是词袋的一种自然扩展。*n*-gram（*n* 元词）是由 *n* 个标记（token）组成的序列。1-gram 就是一个单词（word），又称为**一元词**（unigram）。经过分词（tokenization）之后，计数机制会将单独标记转换为单词计数，或将有重叠的序列作为 *n*-gram 进行计数。例如，句子“Emma knocked on the door”会生成 *n*-gram“Emma knocked”“knocked on”“on the”和“the door”。

n-gram 能够更多地保留文本中的初始序列结构，因此 *n* 元词袋表示法可以表达更丰富的信息。然而，这不是没有代价的。理论上，有 k 个不同的单词，就会有 k^2 个不同的 2-gram（又称**二元词**）。实际上，没有这么多，因为不是每个单词都可以跟在另一个单词后面。尽管如此，*n*-gram（$n>1$）一般来说也会比单词多得多。这意味着 *n* 元词袋是一个更大也更稀疏的特征空间，也意味着 *n* 元词袋需要更强的计算、存储和建模能力。*n* 越大，能表示的信息越丰富，相应的成本也会越高。

为了演示一下 *n*-gram 的数量如何随着 *n* 的增大而增长（见图 3-6），我们计算一下 Yelp 点评数据集中 *n*-gram 的数量。在例 3-1 中，我们使用 Pandas 和 scikit-learn 中的 `CountVectorizer` 转换器计算出了前 10 000 条点评中的 *n*-gram 数量。

例 3-1　计算 *n*-gram

```
>>> import pandas
>>> import json
>>> from sklearn.feature_extraction.text import CountVectorizer

# 加载前10 000条点评
>>> f = open('data/yelp/v6/yelp_academic_dataset_review.json')
>>> js = []
>>> for i in range(10000):
...     js.append(json.loads(f.readline()))
>>> f.close()
>>> review_df = pd.DataFrame(js)

# 创建一元词、二元词和三元词的特征转换器。
# 默认情况下，会忽略单字母词，这非常有实际意义，
# 因为会除去无意义的词。但在这个例子中，
# 出于演示的目的，我们会显式地包含这些词。
```

```
>>> bow_converter = CountVectorizer(token_pattern='(?u)\\b\\w+\\b')
>>> bigram_converter = CountVectorizer(ngram_range=(2,2),
...                                    token_pattern='(?u)\\b\\w+\\b')
>>> trigram_converter = CountVectorizer(ngram_range=(3,3),
...                                     token_pattern='(?u)\\b\\w+\\b')

# 拟合转换器，查看词汇表大小
>>> bow_converter.fit(review_df['text'])
>>> words = bow_converter.get_feature_names()
>>> bigram_converter.fit(review_df['text'])
>>> bigrams = bigram_converter.get_feature_names()
>>> trigram_converter.fit(review_df['text'])
>>> trigrams = trigram_converter.get_feature_names()
>>> print (len(words), len(bigrams), len(trigrams))
26047 346301 847545

# 看一下n-gram
>>> words[:10]
['0', '00', '000', '0002', '00am', '00ish', '00pm', '01', '01am', '02']

>>> bigrams[-10:]
['zucchinis at',
 'zucchinis took',
 'zucchinis we',
 'zuma over',
 'zuppa di',
 'zuppa toscana',
 'zuppe di',
 'zurich and',
 'zz top',
 'à la']

>>> trigrams[:10]
['0 10 definitely',
 '0 2 also',
 '0 25 per',
 '0 3 miles',
 '0 30 a',
 '0 30 everything',
 '0 30 lb',
 '0 35 tip',
 '0 5 curry',
 '0 5 pork']
```

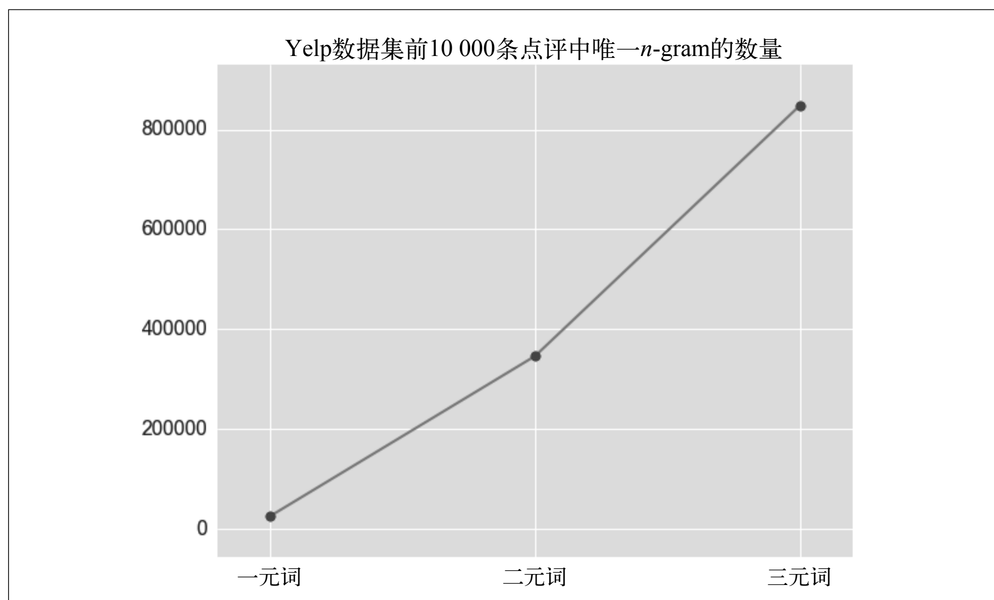


图 3-6：Yelp 数据集前 10 000 条点评中唯一 *n*-gram 的数量

3.2 使用过滤获取清洁特征

如何通过单词将文本中的信号和噪声准确地区分开呢？使用过滤，那些通过原始分词和计数来生成单词或 *n*-gram 列表的技术将变得更有用。下面要介绍的短语检测可以被看作一种特别的二元词过滤器。除此之外，我们还会介绍几种其他的过滤方法。

3.2.1 停用词

分类和提取通常不要求对文本进行深入的理解。例如，在句子“Emma knocked on the door”中，单词“on”和“the”并不能改变这个句子是关于一个人和一扇门这样的事实。在像分类这样的粗粒度任务中，代词、冠词和介词没有什么价值。但在情感分析中，情况就完全不同了，它需要对语义进行细粒度的深刻理解。

Python 中通用的 NLP 包 NLTK 中包含了一个由语言学家定义的停用词列表，适用于多种语言。（你需要先安装 NLTK，并运行 `nltk.download()` 来获取完整功能。）在网上也可以找到各种停用词列表。例如，下面是英语停用词列表中的一些词：

```
a, about, above, am, an, been, didn't, couldn't, i'd, i'll, itself, let's, myself,
our, they, through, when's, whom, ...
```

注意，这个列表中包含撇号，而且单词是小写的。如果想直接使用这个列表，分词过程就不能忽略撇号，而且要将单词转换为小写。

3.2.2 基于频率的过滤

停用词列表是一种剔除形成无意义特征的单词的方法。还有一些更具统计性的方法可以找出这些没有实际意义的单词。在搭配提取方法中，有一些依赖人工定义的方法，也有一些运用统计学的方法。在单词过滤中，我们可以应用同样的思想，也可以使用频率统计。

1. 高频词

频率统计是一种非常强大的过滤技术，既可以过滤语料库专用的常见单词，也可以过滤通用的停用词。例如，短语“New York Times”以及其中的每个单词在纽约时报注释语料库数据集（New York Times Annotated Corpus dataset）中都频繁出现。同样，单词“house”频繁出现在英国议会演讲语料库（Hansard corpus）中的短语“House of Commons”中，这个语料库中的加拿大议会辩论数据集常用于统计机器翻译，因为它包括所有文件的英文和法文版本。这些单词一般来说是有意义的，但在特定语料库中则不然。典型的停用词列表会包括通用的停用词，但不包括语料库专用的停用词。

检查一下出现频率最高的单词，可以发现文本解析时的问题，并能标记出那些碰巧在语料库中出现多次的通常有用的词。举例来说，表 3-1 列出了 Yelp 点评数据集中出现频率最高的 40 个词。这里的频率指的是包含这个词的文档（点评）数，并不是它在一篇文档中出现的次数。正如我们所看到的，这个列表中有很多停用词。我们还有一些意外的发现，列表中有“s”和“t”，这是因为我们使用了撇号作为分词的分隔符，于是像“Mary’s”或“didn’t”这样的词就被解析为“Mary s”和“didn t”。此外，“good”“food”和“great”都出现在了大约三分之一的点评中，但我们要保留它们，因为它们在像情感分析和商家分类这样的任务中是非常有用的。

表3-1：Yelp点评数据集中出现频率最高的词

排名	词	文档频率	排名	词	文档频率
1	the	1 416 058	13	but	822 313
2	and	1 381 324	14	my	786 595
3	a	1 263 126	15	that	777 045
4	i	1 230 214	16	with	775 044
5	to	1 196 238	17	on	735 419
6	it	1 027 835	18	they	720 994
7	of	1 025 638	19	you	701 015
8	for	993 430	20	have	692 749
9	is	988 547	21	t	684 049
10	in	961 518	22	not	649 824
11	was	929 703	23	s	626 764
12	this	844 824	24	had	620 284

（续）

排名	词	文档频率	排名	词	文档频率
25	so	608 061	33	great	520 634
26	place	601 918	34	were	516 685
27	good	598 393	35	there	510 897
28	at	596 317	36	here	481 542
29	are	585 548	37	all	478 490
30	food	562 332	38	if	475 175
31	be	543 588	39	very	460 796
32	we	537 133	40	out	460 452

实际上，频率统计有助于将基于频率的过滤技术与停用词列表结合起来，但有一个问题，就是如何划分二者的界限。遗憾的是，这个问题没有统一的答案。多数情况下，需要人为地确定这个界限，而且要随着数据集的变化而不断调整。

2. 罕见词

根据任务的不同，有时还需要过滤掉罕见词。罕见词可能是真正的生僻词，也可能是拼写错误的普通词。对于统计模型来说，只在一两篇文档中出现的词更像是噪声，而不是有用信息。例如，假设我们的任务是基于 Yelp 点评数据对商家进行分类，而且只有一条点评中包含“gobbledygook”这个词。那么基于这么一个词，我们怎么能辨别出这个商家是餐馆、美发沙龙，还是酒吧呢？在这种情况下，即使我们知道这个商家是个酒吧，如果将包含“gobbledygook”这个词的其他点评划分为酒吧，也很可能是个错误。

罕见词不仅无法作为预测的凭据，还会增加计算上的开销。Yelp 点评数据集中有 160 万条点评数据，包括 357 481 个单词（根据空格和标点符号进行分词），其中有 189 915 个单词只出现在一条点评中，有 41 162 个单词出现在两条点评中。词汇表中 60% 以上的词都是罕见词。这就是所谓的**重尾分布**，在实际数据中这种分布屡见不鲜。很多统计机器学习模型的训练时间是随着特征数量线性增长的，但有些模型则是平方增长或更糟糕。罕见词带来了很大的计算和存储成本，却收效甚微。

在单词数量统计的基础上，我们可以轻松地识别并清理罕见词。或者，也可以将罕见词的数量累加到一个特殊的垃圾箱内，作为一个附加特征。图 3-7 使用一个简短文档演示了这种方法，文档中有一些常见词，以及两个罕见词“gobbledygook”和“zylophant”。常见词使用它们本身的计数，可以进一步通过停用词列表或其他基于频率的方法进行过滤。对于罕见词，则不做区分，统一放到垃圾箱特征中。

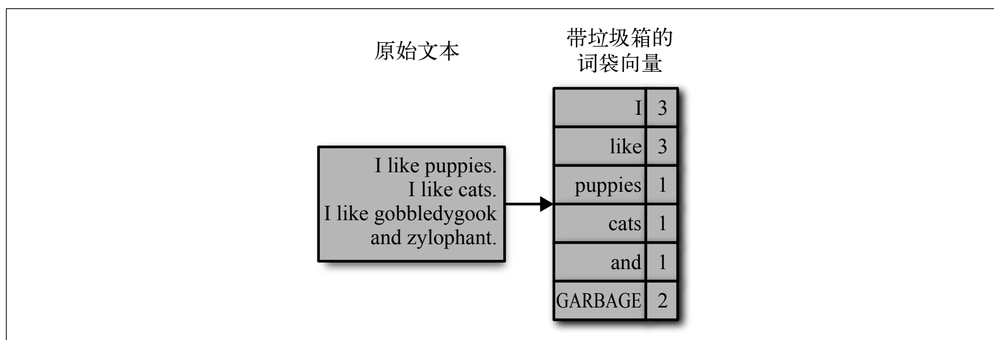


图 3-7：带垃圾箱的词袋特征向量

因为在对整个语料库进行计数统计之前，我们不知道哪些词是罕见的，所以垃圾箱特征只能在后处理阶段进行收集。

既然本书讲述的是特征工程，我们的关注点肯定在于特征。但罕见性这个概念也可以应用在数据点上。如果一个文本文档非常短，那么它很可能不会包含什么有价值的信息，在训练模型时不应该使用它。但是，在应用这条原则时一定要小心。Wikipedia dump 语料库中有很多页还是未完成状态，过滤掉这些页应该是很安全的。推文则是另一种情况，它天生简短，需要专门的特征化和建模技巧。

3.2.3　词干提取

文本的简单解析有一个问题，就是同一个单词的各种变体会被视为不同的词而分别计数。例如，“flower”和“flowers”在技术上是两个不同的标记，“swimmer”“swimming”和“swim”也是一样的情况，尽管它们的含义非常相近。如果这些不同变体能映射为同一单词，那文本解析的效果会更好。

词干提取是一种将每个单词转换为语言学中的基本词干形式的 NLP 技术。词干提取有多种方法，有的基于语言学规则，有的基于统计观测。有一种算法子类综合了词性标注和语言规则，这种处理过程称为词形还原。

多数词干提取工具都将英语作为重点，但针对其他语言的工具也在蓬勃发展。Porter stemmer 是应用最为广泛的免费英语词干提取工具。虽然最初的程序是用 ANSI C 开发的，但有很多程序包对其进行了包装，提供了各种语言接口。

下面是一个通过 Python 的 NLTK 包运行 Porter stemmer 的例子。正如你看到的，它适用于很多情况，但不是万能的。“goes”被映射到了“goe”，而“go”被映射到了它本身。

```
>>> import nltk
>>> stemmer = nltk.stem.porter.PorterStemmer()
>>> stemmer.stem('flowers')
```

```
u'flower'
>>> stemmer.stem('zeroes')
u'zero'
>>> stemmer.stem('stemmer')
u'stem'
>>> stemmer.stem('sixties')
u'sixti'
>>> stemmer.stem('sixty')
u'sixty'
>>> stemmer.stem('goes')
u'goe'
>>> stemmer.stem('go')
u'go'
```

词干提取确实有一些计算成本，最终的收益能否超过成本要视具体应用而定。值得注意的是，使用词干提取可能得不偿失。“new”和“news”具有非常不同的意义，但都会被提取为“new”。同样的例子还有不少。基于这个原因，词干提取并不是非做不可。

3.3 意义的单位：从单词、*n*元词到短语

词袋的概念通俗易懂，但计算机怎么知道什么是一个单词呢？一个文本文档的数字化表示就是一个字符串，也就是一个字符序列。我们还会遇到一些半结构化文档，比如 JSON 字符串或 HTML 页面。但即使添加了标记和结构，文本的基本单位还是字符串。我们如何将字符串转换为一个单词序列呢？这就需要文本的**解析**和**分词**技术，下面就来讨论一下。

3.3.1 解析与分词

当字符串不只包含纯文本时，解析就是必须的。例如，如果原始数据是网页、电子邮件或某种日志，那么其中就含有其他结构。我们需要确定如何处理标记、头部和尾部，以及日志中我们不感兴趣的部分。如果文档是个网页，那么解析程序还需要处理 URL。如果文档是封电子邮件，那么像发件人、收件人和标题这些域都需要特殊处理，否则这些头信息在最终计数中就会和普通词一样，也就失去作用了。

经过简单解析之后，就可以对文档的纯文本部分进行分词了，这会将字符串——一个字符序列——转换为一个标记序列。每个标记都可以作为一个单词来计数。分词程序需要知道哪些字符表示一个标记已经结束而且另一个标记已经开始了。空格通常是一个非常好的分隔符，标点符号也是一样。如果文本中包含推文内容，那么井号（#）就不应该被用作分隔符（又称**定界符**）了。

有时候，分析需要在句子上而不是整个文档上进行。例如，*n* 元词（*n*-gram）是单词概念的一个推广，它就不能超过句子的边界。像 word2vec 这样比较复杂的文本特征化方法也是工作在句子或段落上的。在这种情况下，我们需要先将文档解析为句子，然后再对每个句子进行分词，得到单词。



字符串对象：并不像你看到的那么简单

字符串对象有多种编码方式，比如 ASCII 或 Unicode。纯英文文本可以用 ASCII 进行编码，但多数其他语言需要使用 Unicode。如果文档中包括非 ASCII 字符，就要确保分词程序可以处理相应的编码方式。否则，分词结果就会出现错误。

3.3.2 通过搭配提取进行短语检测

通过标记序列可以立刻得到单词和 *n* 元词列表。但是，从语义上说，我们更习惯于理解短语，而不是 *n* 元词。在计算机自然语言处理（NLP）中，有用短语的概念被称为**搭配**（collocation）。用 Manning 和 Schütze（1999: 151）的话来说："搭配是一种表达方式，它由两个或两个以上的单词组成，并对应于某种约定俗成的事物说明。"

搭配能表达的意义比组成它的各个单词的总和还要多。例如，"strong tea"的意义绝对不止"great physical strength"和"tea"，因此可以认为它是个搭配。另一方面，短语"cute puppy"的意义则就是两个单词"cute"和"puppy"之和，因此我们认为它不是个搭配。

搭配不一定是个连贯的序列。例如，可以认为句子"Emma knocked on the door"包含搭配"knock door"，因此，不是所有的搭配都是 *n* 元词。反之，也不是所有 *n* 元词都一定是有意义的搭配。

因为搭配的意义比组成它的各个单词的总和要多，所以单词计数不能恰当地表示出它的意义。这时用词袋来表示就力不从心了，用 *n* 元词袋表示也有问题，因为 *n* 元词袋中有太多无意义的序列（比如 *n* 元词袋示例中的"this is"），有意义的序列（如 knock door）则不够多。

搭配非常适合用作特征，但我们如何从文本中发现并提取它们呢？一种方法是进行预定义。如果我们竭尽全力，就可以做出一个包含多种语言惯用语的综合性列表，然后就可以仔细检查文本，找出所有的匹配。这样做成本很高，但肯定有效。如果语料库只局限于一个特定领域，而且有很多专业的词语，那么这应该是最好的方法。但这个列表需要大量人工干预，并且需要随着语料库的发展而频繁更新。对于推文、博客和新闻文章来说，这种方法就不太现实了。

由于过去 20 年中统计自然语言处理的出现和发展，人们已经越来越愿意使用统计方法来找出短语。与建立一个短语与惯用语的固定列表不同，统计性的搭配提取方法可以根据不断发展变化的数据来找出当前的流行用语。

1. 基于频率的方法

一种简单方法是查看那些出现频率最高的 *n* 元词。这种方法的问题是最常出现的词不一定是最有用的。表 3-2 中列出了 Yelp 点评数据集中最常见的二元词。如你所见，通过文档计数得出的最常见的 10 个二元词都是平淡无奇的词语，没有多少意义。

表3-2：Yelp点评数据集中出现频率最高的二元词

二元词	文档计数
of the	450 849
and the	426 346
in the	397 821
it was	396 713
this place	344 800
it s 3	41 090
and i	332 415
on the	325 044
i was	285 012
for the	276 946

2. 用于搭配提取的假设检验

简单计数是一种太简陋的测量方式，我们必须找到更加聪明的统计手段，以便轻松地提取出有意义的短语。其中的关键是要回答一个问题：对于两个经常同时出现的词，它们同时出现的频率是否远高于由于偶然才同时出现的频率。能回答这个问题的统计学手段就是**假设检验**。

假设检验是一种将带噪声的数据归结为"是"或"否"两种答案的方法。它可以对从随机分布中抽取出的样本数据进行建模。随机性意味着我们从来不会百分之百地确定答案，总有出现异常值的可能。所以，答案中要附带一个概率。

例如，假设检验的结果可以是"这两个数据集有 95% 的概率来自于同一分布"。关于假设检验的详细介绍，可以看一下可汗学院关于假设检验和 p 值的教程。

在搭配提取方面，近年来提出了很多相关的假设检验方法，其中最成功的方法之一是基于似然比的检验（Dunning, 1993）。对于一对给定的单词，该方法在观测数据集上检验两个假设。第一个假设（原假设）认为单词 1 的出现与单词 2 无关，换种说法就是看到单词 1 对于能否看到单词 2 没有影响。第二个假设（备择假设）则认为看到单词 1 会改变看到单词 2 的可能。如果我们接受了备择假设，就意味着这两个单词可以组成一个常见短语。因此，用于短语检测（即搭配提取）的似然比检验会提出以下问题：在一个特定的文本语料库中，我们可以建立两个模型，其中一个模型中的单词出现频率是彼此不相关的，另一个模型中的单词出现频率则彼此相关，那么在语料库中实际观测到的单词出现频率更可能是哪个模型中生成的频率呢？

听起来很拗口，我们用数学公式把它表述得更清晰一些。（数学是一种非常好的表达方式，可以将事物表达得精确而又简洁，但它需要一种与自然语言完全不同的解析方法。）

我们可以将原假设 H_{null}（不相关）表述为 $P(w_2 \mid w_1) = P(w_2 \mid \text{not } w_1)$，将备择假设 $H_{alternate}$（相关）表述为 $P(w_2 \mid w_1) \neq (w_2 \mid \text{not } w_1)$。

最终统计量为以下两个函数的比值的对数：

$$\log \lambda = \log \frac{L(\text{Data}; H_{null})}{L(\text{Data}; H_{alternate})}$$

似然函数 $L(\text{Data}; H)$ 表示在使用相关模型或不相关模型时，得到一对单词在数据集中出现频率的概率。为了计算出这个概率，还必须做出另外一个假设，即数据是如何生成的。最简单的数据生成模型是二项式模型，对于数据集中的每个单词，我们都掷一次硬币，如果硬币正面向上，我们就插入一个特定单词，否则就插入某个其他单词。在这种策略下，这个特定单词的出现次数就服从**二项式分布**。二项式分布完全由单词的总数、感兴趣单词的出现次数和硬币正面向上的概率来决定。

通过似然比检验这种分析方法来检测常见短语的算法如下。

(1) 计算出所有单词的出现概率：$P(w)$。
(2) 对所有的唯一二元词，计算出成对单词出现的条件概率：$P(w_2 \mid w_1)$。
(3) 对所有的唯一二元词，计算出似然比 $\log \lambda$。
(4) 按照似然比为二元词排序。
(5) 将似然比最小的二元词作为特征。



理解似然比检验

似然比检验的关键在于，它比较的不是概率参数本身，而是在这些参数（以及预设的数据生成模型）之下得到实际观测数据的概率。似然是统计学习中的一个核心概念，刚学习时会觉得它难以理解，但是一旦理解了其中的逻辑，就会觉得它很直观易懂。

还有一种基于点间互信息的统计方法，但它对罕见词极为敏感，而现实世界的文本语料库中总是免不了罕见词的。因此，这种方法并不常用，这里也就不介绍了。

请注意，所有用于搭配提取的统计方法都要对一个候选短语列表进行过滤，不管它使用的方法是简单词频统计、假设检验，还是点间互信息。生成这种列表的最简单实惠的方法是对 n 元词进行计数。可以生成不连贯的序列，但计算成本比较高。在实际分析中，即使是对于连贯的 n 元词，人们也很少使用二元词和三元词之外的 n 元词，因为即使在过滤之后，它们的数量仍然太多。要生成更长的短语，需要另外的方法，比如文本分块，或结合词性标注来使用。

3. 文本分块和词性标注

文本分块要比找出 *n* 元词复杂一些，复杂之处在于，它要使用基于规则的模型并基于词性生成标记序列。

举例来说，或许我们最感兴趣的是找出一个问题中的所有名词短语，即这个问题中的实体（在下面的例子中是文本标题）。为了找出这些短语，我们先切分出所有带词性的单词，然后检查这些标记的邻近词，找出按词性组合的词组，这些词组又称为"块"。将单词映射到词性的模型通常与特定的语言有关。一些开源的 Python 程序库（比如 NLTK、spaCy 和 TextBlob）中带有适用于多种语言的模型。

为了尽量直观地演示一下 Python 中的几个程序库是如何使用词性标注来进行文本分块的，我们再用一次 Yelp 点评数据集。在例 3-2 中，我们使用 spaCy 和 TextBlob 通过判断词性来找出名词短语。

例 3-2　词性标注和文本分块

```
>>> import pandas as pd
>>> import json

# 加载前10条点评
>>> f = open('data/yelp/v6/yelp_academic_dataset_review.json')
>>> js = []
>>> for i in range(10):
...     js.append(json.loads(f.readline()))
>>> f.close()
>>> review_df = pd.DataFrame(js)

# 首先使用spaCy中的函数
>>> import spacy
# 预先加载语言模型
>>> nlp = spacy.load('en')

# 我们可以创建一个spaCy nlp变量的Pandas序列
>>> doc_df = review_df['text'].apply(nlp)

# spaCy可以使用(.pos_)提供细粒度的词性，
# 使用(.tag_)提供粗粒度的词性
>>> for doc in doc_df[4]:
...     print([doc.text, doc.pos_, doc.tag_])

Got VERB VBP
a DET DT
letter NOUN NN
in ADP IN
the DET DT
mail NOUN NN
last ADJ JJ
week NOUN NN
that ADJ WDT
said VERB VBD
```

```
Dr. PROPN NNP
Goldberg PROPN NNP
is VERB VBZ
moving VERB VBG
to ADP IN
Arizona PROPN NNP
to PART TO
take VERB VB
a DET DT
new ADJ JJ
position NOUN NN
there ADV RB
in ADP IN
June PROPN NNP
. PUNCT .
  SPACE SP
He PRON PRP
will VERB MD
be VERB VB
missed VERB VBN
very ADV RB
much ADV RB
. PUNCT .


SPACE SP
I PRON PRP
think VERB VBP
finding VERB VBG
a DET DT
new ADJ JJ
doctor NOUN NN
in ADP IN
NYC PROPN NNP
that ADP IN
you PRON PRP
actually ADV RB
like INTJ UH
might VERB MD
almost ADV RB
be VERB VB
as ADV RB
awful ADJ JJ
as ADP IN
trying VERB VBG
to PART TO
find VERB VB
a DET DT
date NOUN NN
! PUNCT .

# spaCy还可以进行基本的名词分块
>>> print([chunk for chunk in doc_df[4].noun_chunks])
[a letter, the mail, Dr. Goldberg, Arizona, a new position, June, He, I,
a new doctor, NYC, you, a date]
```

```
#####
# 我们还可以使用TextBlob实现同样的特征转换
from textblob import TextBlob

# TextBlob中的默认标记器使用PatternTagger，在这个例子中是没有问题的。
# 你还可以指定使用NLTK标记器，它对于不完整的句子效果更好。
>>> blob_df = review_df['text'].apply(TextBlob)

>>> blob_df[4].tags
[('Got', 'NNP'),
('a', 'DT'),
('letter', 'NN'),
('in', 'IN'),
('the', 'DT'),
('mail', 'NN'),
('last', 'JJ'),
('week', 'NN'),
('that', 'WDT'),
('said', 'VBD'),
('Dr.', 'NNP'),
('Goldberg', 'NNP'),
('is', 'VBZ'),
('moving', 'VBG'),
('to', 'TO'),
('Arizona', 'NNP'),
('to', 'TO'),
('take', 'VB'),
('a', 'DT'),
('new', 'JJ'),
('position', 'NN'),
('there', 'RB'),
('in', 'IN'),
('June', 'NNP'),
('He', 'PRP'),
('will', 'MD'),
('be', 'VB'),
('missed', 'VBN'),
('very', 'RB'),
('much', 'JJ'),
('I', 'PRP'),
('think', 'VBP'),
('finding', 'VBG'),
('a', 'DT'),
('new', 'JJ'),
('doctor', 'NN'),
('in', 'IN'),
('NYC', 'NNP'),
('that', 'IN'),
('you', 'PRP'),
('actually', 'RB'),
('like', 'IN'),
('might', 'MD'),
('almost', 'RB'),
('be', 'VB'),
```

```
    ('as', 'RB'),
    ('awful', 'JJ'),
    ('as', 'IN'),
    ('trying', 'VBG'),
    ('to', 'TO'),
    ('find', 'VB'),
    ('a', 'DT'),
    ('date', 'NN')]

>>> print([np for np in blob_df[4].noun_phrases])
['got', 'goldberg', 'arizona', 'new position', 'june', 'new doctor', 'nyc']
```

可见，通过不同程序库找出的名词短语并不完全一样。spaCy 找出的短语中包括英文中的一些普通词，如“a”和“the”，而 TextBlob 则去掉了这些词。由此可知，不同程序库中认定名词短语的规则引擎是有区别的。你还可以编写自己的词性关系来定义要搜寻的文本块。你可以参考（Bird 等，2009）来从头开始研究使用 Python 进行文本分块的方法。

3.4 小结

词袋表示法简单易懂，容易计算，并对分类和搜索任务非常有效。但有时单个单词还是太简单了，无法表述出文本中的某些信息。为了解决这个问题，我们要求助于更长的序列。*n* 元词袋是词袋的一种自然推广，它的概念非常好理解，计算起来也和词袋一样容易。

n 元词袋可以生成大量互不相同的 *n* 元词，它增加了特征存储成本，在模型训练和预测阶段也需要更多计算能力。对于同样数量的数据点，*n* 元词袋使得特征空间的维度大大增加。因此，数据变得特别稀疏。*n* 越大，存储和计算的成本就越高，数据也越稀疏。基于这些原因，更长的 *n* 元词并不是总能提高模型的准确率或带来其他方面的性能改善。通常只使用二元词和三元词，很少使用更长的 *n* 元词。

要解决稀疏性和成本增加的问题，一种方法是对 *n* 元词进行过滤，只保留那些最有意义的短语。这就是搭配提取的目标。理论上，搭配（或短语）可以形成文本中不连贯的标记序列，但实际上，找出不连贯的短语需要非常高的计算成本，而且收效甚微。所以，搭配提取通常从一个备选二元词列表开始，然后使用统计方法对其进行过滤。

所有这些方法都是将一个文本标记序列转换为一个与之无关的计数集合。相对于单词序列，集合中的结构很少，它们可以生成扁平的特征向量。

本章蜻蜓点水式地介绍了一下简单的文本特征化技术。这些技术将一段带有丰富语义结构的自然语言文本转换为简单的扁平向量。我们讨论了一些常用的能使向量变得更加整洁的过滤技术，还介绍了 *n* 元词和搭配提取，将它们作为一种向扁平向量中添加一点结构的方法。下一章将更加详细地介绍另一种常用的文本特征化方法，称为 **tf-idf**。随后的章节中会讨论更多向扁平向量中添加结构的方法。

3.5 参考文献

Bird, Steven, Ewan Klein, and Edward Loper. Natural Language Processing with Python [M]. Sebastopol, CA: O’Reilly Media, 2009.

Dunning, Ted. Accurate Methods for the Statistics of Surprise and Coincidence [J]. ACM Journal of Computational Linguistics, special issue on using large corpora 19:1 (1993): 61–74.

Khan Academy. Hypothesis Testing and p-Values [EB/OL]. https://www.khanacademy.org/math/probability/statistics-inferential/hypothesis-testing/v/hypothesis-testing-and-p-values.

Manning, Christopher D. and Hinrich Schütze. Foundations of Statistical Natural Language Processing [M]. Cambridge, MA: MIT Press, 1999.

第 4 章

特征缩放的效果：从词袋到tf-idf

词袋表示法简单易行，但远非完美。如果我们无差别地对所有单词计数，那么有些单词会被过分强调，这是根本不必要的。回想一下第 3 章中 Emma 和乌鸦的例子，我们希望有一种能够强调两个主要角色的文档表示方法。单词“Emma”和“raven”都出现了 3 次，但“the”居然出现了 8 次，“and”也出现了 5 次，“it”和“was”都出现了 4 次。仅通过简单的词频计数，无法突显出主要角色，这是这种方法的问题所在。

能挑选出像“magnificently”“gleamed”“intimidated”“tentatively”和“reigned”这样的单词也是很好的，因为它们有助于确定该段文字的整体基调。它们能体现出情感，这对数据科学家来说是非常宝贵的信息。所以，理想情况下，我们需要那种能强调出**有意义**的单词的表示方法。

4.1　tf-idf：词袋的一种简单扩展

tf-idf 是在词袋方法基础上的一种简单扩展，它表示**词频－逆文档频率**。tf-idf 计算的不是数据集中每个单词在每个文档中的原本计数，而是一个归一化的计数，其中每个单词的计数要除以这个单词出现在其中的文档数量。即：

bow(*w*, *d*) = 单词 *w* 在文档 *d* 中出现的次数
tf-idf(*w*, *d*) = bow(*w*, *d*) * *N* / (单词 *w* 出现在其中的文档数量)

N 是数据集中的文档总数。分数 *N* / (单词 *w* 出现在其中的文档的数量) 就是所谓的**逆文档频率**。如果一个单词出现在很多文档中，那么它的逆文档频率就接近于 1。如果一个单词只出现在少数几个文档中，那么它的逆文档频率就会高得多。

我们也可以使用逆文档频率的对数变换，而不是它的原始形式。对数变换可以将 1 转换为 0，并使大的数值（那些远远大于 1 的值）变小。（随后会有更多介绍。）

如果将 tf-idf 定义为：

tf-idf(*w*, *d*) = bow(*w*, *d*) * log(*N* / 单词 *w* 出现在其中的文档数量)

那么就可以有效地将一个几乎出现在所有单个文档中的单词的计数归零，而一个只出现在少数几个文档中的单词的计数将会被放大。

我们通过几张图片来加深一下理解。图 4-1 是个非常简单的例子，其中有 4 个句子："it is a puppy""it is a cat""it is a kitten"和"that is a dog and this is a pen"。我们在由"puppy""cat"和"is"这 3 个单词构成的特征空间中绘制出这些句子。

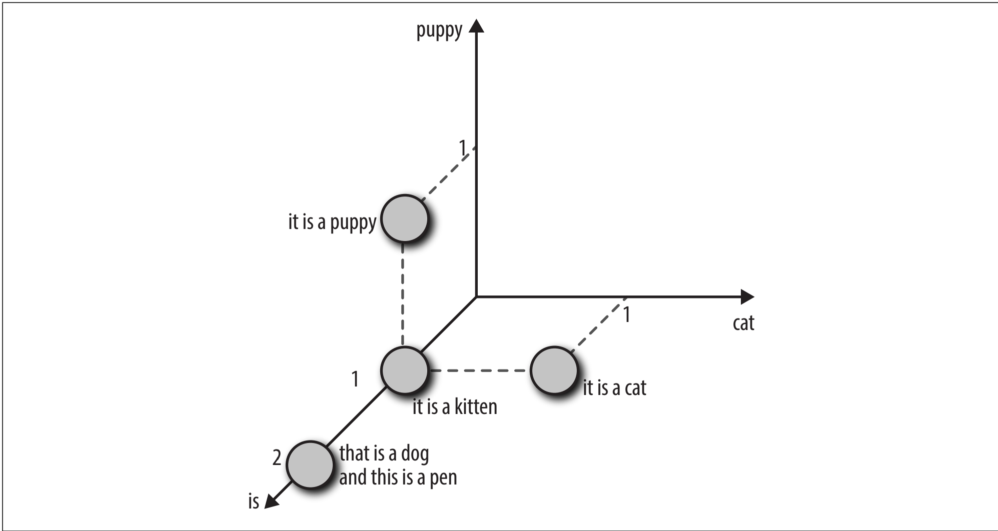


图 4-1：关于狗和猫的 4 个句子

下面看一下 tf-idf 表示法中同样的 4 个句子，我们对逆文档频率使用了对数变换。图 4-2 展示了特征空间中的文档，可以看到，单词"is"作为一个特征被有效地消除了，因为它在这个数据集的所有句子中都出现了。另外，单词"puppy"和"cat"因为在 4 个句子中都只出现了 1 次，所以它们的计数被放大了（log(4) = 1.38... > 1）。因此，tf-idf 使得罕见词更突出，并有效地忽略了常见词。tf-idf 与第 3 章中基于频率的过滤方法密切相关，但相对于设置固定边界阈值的做法，它在数学上显得更加优雅。



tf-idf 的直观理解

tf-idf 突出了罕见词，并有效地忽略了常见词。

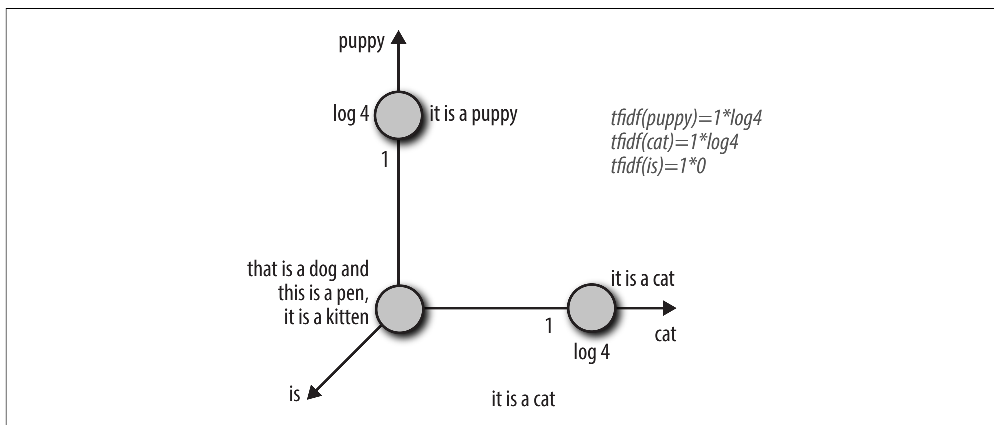


图 4-2：图 4-1 中句子的 tf-idf 表示

4.2 tf-idf方法测试

tf-idf 通过乘以一个常数，对单词计数特征进行了转换。因此，它是一种**特征缩放**方法，我们在第 2 章中介绍过这个概念。特征缩放的实际效果如何呢？让我们在一个简单的文本分类任务中比较一下缩放特征和未缩放特征的效果。是时候做一些编码了！

在例 4-1 中，我们再次使用了 Yelp 点评数据集。Yelp 数据竞赛第 6 轮的数据集中有差不多 160 万条商家点评数据，这些商家分布在美国的 6 个城市中。

例 4-1　使用 Python 加载并清理 Yelp 点评数据集

```
>>> import json
>>> import pandas as pd

# 加载Yelp商家数据
>>> biz_f = open('yelp_academic_dataset_business.json')
>>> biz_df = pd.DataFrame([json.loads(x) for x in biz_f.readlines()])
>>> biz_f.close()

# 加载Yelp点评数据
>>> review_file = open('yelp_academic_dataset_review.json')
>>> review_df = pd.DataFrame([json.loads(x) for x in review_file.readlines()])
>>> review_file.close()

# 选取出夜店和餐馆
>>> two_biz = biz_df[biz_df.apply(lambda x: 'Nightlife' in x['categories'] or
...                                         'Restaurants' in x['categories'],
...                                  axis=1)]

# 与点评数据连接，得到两种类型商家的所有点评
>>> twobiz_reviews = two_biz.merge(review_df, on='business_id', how='inner')

# 去除我们不需要的特征
```

```
>>> twobiz_reviews = twobiz_reviews[['business_id',
...                                  'name',
...                                  'stars_y',
...                                  'text',
...                                  'categories']]

# 创建目标列——夜店类型的商家为True，否则为False
>>> two_biz_reviews['target'] = \
...     twobiz_reviews.apply(lambda x: 'Nightlife' in x['categories'],
...                          axis=1)
```

4.2.1　创建分类数据集

下面看看能否通过点评数据区分出一个商家是餐馆还是夜店。为了节省训练时间，我们可以取点评数据的一个子集。在这个例子中，两类商家的点评数量相差很大，这称为**类别不平衡数据集**。不平衡数据集的建模有些问题，因为模型会将大部分努力用于拟合优势类别。因为在两个类别中我们都有很多数据，所以解决这个问题的一种好的做法是对优势类别（餐馆）进行下采样，使它的数量与劣势类别（夜店）基本相当。下面是一个示例流程。

(1) 对夜店点评数据进行 10% 的随机抽样，对餐馆点评数据进行 2.1% 的随机抽样（选择这样的比例可以使两个类别的抽样数据基本相当）。
(2) 按照 70/30 的比例将这个数据集划分为训练集和测试集。在这个例子中，训练集有 29 264 条点评数据，测试集有 12 542 条点评数据。
(3) 训练数据包含 46 924 个唯一单词，这就是词袋表示法的特征数量。

例 4-2 给出了具体做法。

例 4-2　创建平衡的分类数据集

```
# 创建一个类别平衡的子样本，供练习使用
>>> nightlife = \
...   twobiz_reviews[twobiz_reviews.apply(lambda x: 'Nightlife' in x['categories'],
...                                       axis=1)]
>>> restaurants = \
...   twobiz_reviews[twobiz_reviews.apply(lambda x: 'Restaurants' in x['categories'],
...                                       axis=1)]
>>> nightlife_subset = nightlife.sample(frac=0.1, random_state=123)
>>> restaurant_subset = restaurants.sample(frac=0.021, random_state=123)
>>> combined = pd.concat([nightlife_subset, restaurant_subset])

# 划分训练集和测试集
>>> training_data, test_data = modsel.train_test_split(combined,
...                                                    train_size=0.7,
...                                                    random_state=123)
>>> training_data.shape
(29264, 5)
>>> test_data.shape
(12542, 5)
```

4.2.2 使用tf-idf变换来缩放词袋

这个练习的目的是比较一下词袋、tf-idf 和 ℓ^2 归一化在线性分类问题中的效果。请注意，做完 tf-idf 后再进行 ℓ^2 归一化等同于只做 ℓ^2 归一化。所以，我们只需要测试三组特征：词袋、tf-idf，以及词袋基础上的 ℓ^2 归一化。

在例 4-3 中，我们使用 scikit-learn 中的 `CountVectorizer` 将点评文本转换为词袋。所有文本特征化方法都依赖于一个分词器，它是一个能将文本字符串转换为标记（单词）列表的程序模块。在这个例子中，scikit-learn 使用默认的分词模式，搜索由 2 个或 2 个以上的字母和数字组成的序列，标点符号被当作标记分隔符。

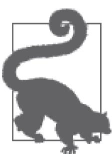
例 4-3　转换特征

```
# 用词袋表示点评文本
>>> bow_transform = text.CountVectorizer()
>>> X_tr_bow = bow_transform.fit_transform(training_data['text'])
>>> X_te_bow = bow_transform.transform(test_data['text'])
>>> len(bow_transform.vocabulary_)
46924

>>> y_tr = training_data['target']
>>> y_te = test_data['target']

# 使用词袋矩阵创建tf-idf表示
>>> tfidf_trfm = text.TfidfTransformer(norm=None)
>>> X_tr_tfidf = tfidf_trfm.fit_transform(X_tr_bow)
>>> X_te_tfidf = tfidf_trfm.transform(X_te_bow)

# 仅出于练习的目的，对词袋表示进行l2归一化
>>> X_tr_l2 = preproc.normalize(X_tr_bow, axis=0)
>>> X_te_l2 = preproc.normalize(X_te_bow, axis=0)
```



测试集上的特征缩放

特征缩放的微妙之处在于，它要求我们知道一些实际中我们很可能不知道的特征统计量，比如均值、方差、文档频率、ℓ^2 范数，等等。为了计算出 tf-idf 表示，我们必须基于**训练**数据计算出逆文档频率，并用这些统计量既缩放训练数据也缩放测试数据。在 scikit-learn 中，在训练数据上拟合特征转换器相当于收集相关统计量。然后可以将拟合好的特征转换器应用到测试数据上。

当使用训练数据统计量来缩放测试数据时，结果看上去会有点奇怪。测试集上的 min-max 缩放不会严格地映射到 0 和 1 之间。ℓ^2 范数、均值和方差统计量看上去也有点不对劲。这是仅次于缺失数据的一个问题。举例来说，如果测试集中含有没出现在训练集中的单词，那么这些新单词就没有可用的文档频率。通常的解决方法是直接将测试集中的新单词删除。这看上去有点不负责任，但模型——通过训练集得出的模型——真的不知道如何处理这些单词。另外一种稍微温柔一点的做法是，显式地设置一个“垃圾词”特征，将所有低

频词都映射到这个特征，包括训练集中的低频词。这种方法在 3.2.2 节的“罕见词”部分讨论过。

4.2.3 使用逻辑回归进行分类

逻辑回归是一种简单的线性分类器。正因为其简单，所以非常适合充当首次试验的分类器。它对输入特征进行加权组合，然后传递给一个 **S 形函数**，这个函数可以平滑地将任何实数映射为 0 和 1 之间的一个值，即将实数输入 *x* 转换为 0 和 1 之间的一个值。它有一组参数 *w*，表示中点（0.5）附近的增长斜率。截距项 *b* 表示函数输出跨过中点时的输入值。如果 S 形函数的输出大于 0.5，逻辑回归分类器就预测一个正分类，否则就预测一个负分类。通过改变 *w* 和 *b*，我们可以控制在哪里发生决策变化，以及决策对在该点附近发生的输入变化做出反应的速度。

图 4-3 演示了 S 形函数。

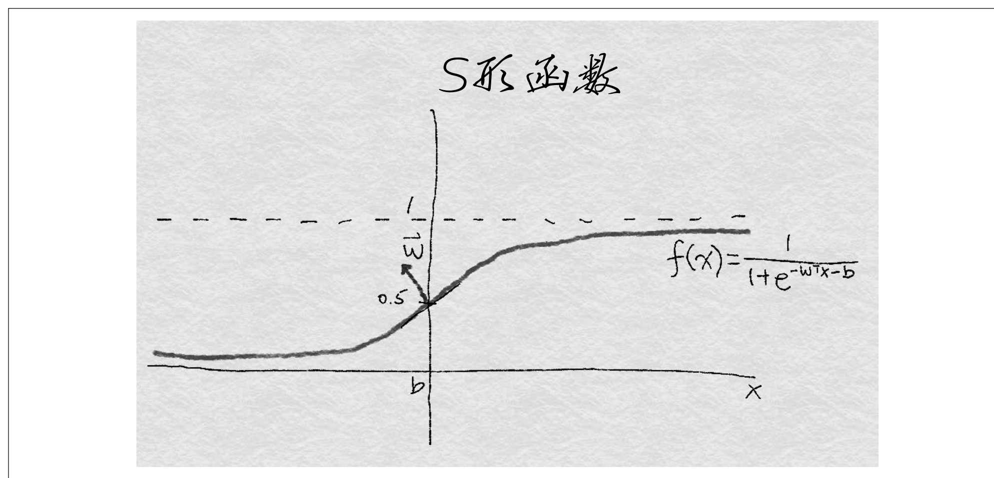


图 4-3：S 形函数示意图

下面我们在不同的特征集合上建立几个简单逻辑回归分类器，并看看效果（见例 4-4）。

例 4-4 使用默认参数训练逻辑回归分类器

```
>>> def simple_logistic_classify(X_tr, y_tr, X_test, y_test, description):
...     ### 辅助函数，用来训练逻辑回归分类器，并在测试数据上进行评分。
...     m = LogisticRegression().fit(X_tr, y_tr)
...     s = m.score(X_test, y_test)
...     print ('Test score with', description, 'features:', s)
...     return m

>>> m1 = simple_logistic_classify(X_tr_bow, y_tr, X_te_bow, y_te, 'bow')
>>> m2 = simple_logistic_classify(X_tr_l2, y_tr, X_te_l2, y_te, 'l2-normalized')
>>> m3 = simple_logistic_classify(X_tr_tfidf, y_tr, X_te_tfidf, y_te, 'tf-idf')
```

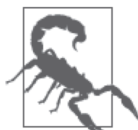
```
Test score with bow features: 0.775873066497
Test score with l2-normalized features: 0.763514590974
Test score with tf-idf features: 0.743182905438
```

事与愿违，结果显示准确率最高的分类器使用的是词袋特征，这真是出乎意料。实际上，出现这种情况的原因在于分类器没有很好地“调优”，这是在比较分类器时经常犯的错误。

4.2.4　使用正则化对逻辑回归进行调优

逻辑回归有些不切实际的功能。当特征数量大于数据点数量时，找出最佳模型这个问题就变得**不确定**了。解决这个问题的一种方法是在训练过程中加入额外的限制条件，这就是**正则化**，本节将讨论它的技术细节。

逻辑回归的大多数具体实现都允许正则化。要使用正则化，必须确定一个正则化参数。正则化参数是一种**超参数**，不能在模型训练过程中自动学习。相反，它们必须根据具体的问题进行调优，并提供给训练算法，这个过程就是超参数调优。（如果想详细了解如何评价机器学习模型，参见 Zheng (2015)。）一种基本的超参数调优方法称为**网格搜索**：先确定一个超参数网格，然后使用调优程序自动搜索，找到网格中的最优超参数设置。找到最优超参数设置之后，你可以使用该设置在整个训练集上训练一个模型，然后使用它在测试集上的表现作为这类模型的最终评价。



重要：比较模型时要对超参数进行调优

当比较模型或特征时，必须对超参数进行调优。软件包的默认设置总是能返回一个模型，但除非软件包能在后台自动调优，否则它很可能基于非最优的超参数设置返回一个非最优模型。分类器性能对超参数设置的敏感度依赖于具体的模型和训练数据的分布。相对来说，逻辑回归对于超参数设置是不太敏感的，但即便这样，找到并使用正确的超参数**范围**还是必要的。否则，如果仅因为超参数调优的原因，一个模型才比另一个模型有优势，就反映不出模型或特征的实际行为了。

即使是最好的能自动调优的软件包，也需要确定搜索的上界和下界，而确定这些界限也需要手动尝试若干次。

在下面的例子中，我们手动设置逻辑回归正则化参数的搜索网格为 {1e-5, 0.001, 0.1, 1, 10, 100}，上界和下界是经过了几次尝试后确定的。每个特征集合的最优超参数设置见表 4-1。

表4-1：逻辑回归的最优超参数设置，使用Yelp点评数据的夜店和餐馆抽样

	ℓ^2正则化
词袋	0.1
ℓ^2 归一化	10
tf-idf	0.001

我们还想测试一下，tf-idf 和词袋之间的准确度差别是否是由噪声造成的。为此，我们使用 *k*-折交叉验证来模拟多个统计独立的数据集，把数据集分成 *k* 折。交叉验证过程会在这些数据子集中迭代进行，使用除一折数据之外的所有数据进行训练，而用保留的那一折数据来验证结果。

通过重采样估计方差

现代统计方法假定基本数据来自于随机分布。对于从这种数据导出的模型，它的性能测量也会受到随机噪声的影响。在这种情况下，好的做法是不止做一次测量，而是基于具有大致相同的统计量的数据集进行多次测量。这样可以获得一个测量结果的置信区间。

k-折交叉验证就是这样一种方法。重采样则是另一种能从同一份基础数据中生成多个小样本的技术。可以参考 Zheng (2015) 了解关于重采样的详细介绍。

scikit-learn 中的 `GridSearchCV` 函数可以执行带交叉验证的网格搜索（见例 4-5）。图 4-4 展示了一张箱线图，给出了使用每种特征集合训练出的模型的准确度测量结果的分布。方框中间的直线表示准确度的中位数，方框本身表示第一四分位数和第三四分位数之间的区域，两侧伸展出去的直线则表示分布的其余部分。

例 4-5　使用网格搜索对逻辑回归进行调优

```
>>> import sklearn.model_selection as modsel

# 确定一个搜索网格，然后对每种特征集合执行5-折网格搜索
>>> param_grid_ = {'C': [1e-5, 1e-3, 1e-1, 1e0, 1e1, 1e2]}

# 为词袋表示法进行分类器调优
>>> bow_search = modsel.GridSearchCV(LogisticRegression(), cv=5,
...                                  param_grid=param_grid_)
>>> bow_search.fit(X_tr_bow, y_tr)

# 为L2-归一化词向量进行分类器调优
>>> l2_search = modsel.GridSearchCV(LogisticRegression(), cv=5,
...                                 param_grid=param_grid_)
>>> l2_search.fit(X_tr_l2, y_tr)

# 为tf-idf进行分类器调优
>>> tfidf_search = modsel.GridSearchCV(LogisticRegression(), cv=5,
...                                    param_grid=param_grid_)
>>> tfidf_search.fit(X_tr_tfidf, y_tr)

# 检查网格搜索的一个输出，看看它是如何运行的
>>> bow_search.cv_results_
{'mean_fit_time': array([  0.43648252,   0.94630651,
          5.64090128,  15.31248307,  31.47010217,  42.44257565]),
'mean_score_time': array([ 0.00080056,  0.00392466,  0.00864897,  0 .00784755,
         0.01192751,  0.0072515 ]),
'mean_test_score': array([ 0.57897075,  0.7518111 ,  0.78283898,  0.77381766,
         0.75515992,  0.73937261]),
```

```
 'mean_train_score': array([ 0.5792185 ,  0.76731652,  0.87697341,  0.94629064,
         0.98357195,  0.99441294]),
 'param_C': masked_array(data = [1e-05 0.001 0.1 1.0 10.0 100.0],
              mask = [False False False False False False],
        fill_value = ?),
 'params': ({'C': 1e-05},
  {'C': 0.001},
  {'C': 0.1},
  {'C': 1.0},
  {'C': 10.0},
  {'C': 100.0}),
 'rank_test_score': array([6, 4, 1, 2, 3, 5]),
 'split0_test_score': array([ 0.58028698,  0.75025624,  0.7799795 ,  0.7726341 ,
         0.75247694,  0.74086095]),
 'split0_train_score': array([ 0.57923964,  0.76860316,  0.87560871,  0.94434003,
         0.9819308 ,  0.99470312]),
 'split1_test_score': array([ 0.5786776 ,  0.74628396,  0.77669571,  0.76627371,
         0 .74867589,  0.73176149]),
 'split1_train_score': array([ 0.57917218,  0.7684849 ,  0.87945837,  0.94822946,
         0.98504976,  0.99538678]),
 'split2_test_score': array([ 0.57816504,  0.75533914,  0.78472578,  0.76832394,
         0.74799248,  0.7356911 ]),
 'split2_train_score': array([ 0.57977019,  0.76613558,  0.87689548,  0.94566657,
         0.98368288,  0.99397719]),
 'split3_test_score': array([ 0.57894737,  0.75051265,  0.78332194,  0.77682843,
         0.75768968,  0.73855092]),
 'split3_train_score': array([ 0.57914745,  0.76678626,  0.87634546,  0.94558346,
         0.98385443,  0.99474628]),
 'split4_test_score': array([ 0.57877649,  0.75666439,  0.78947368,  0.78503076,
         0.76896787,  0.75      ]),
 'split4_train_score': array([ 0.57876303,  0.7665727 ,  0.87655903,  0.94763369,
         0.98334188,  0.99325132]),
 'std_fit_time': array([ 0.03874582,  0.02297261,  1.18862097,  1.83901079,
         4.21516797,  2.93444269]),
 'std_score_time': array([ 0.00160112,  0.00605009,  0.00623053,  0.00698687,
         0.00713112,  0.00570195]),
 'std_test_score': array([ 0.00070799,  0.00375907,  0.00432957,  0.00668246,
         0.00612049]),
 'std_train_score': array([ 0.00032232,  0.00102466,  0.00131222,  0.00143229,
         0.00100223,  0.00073252])}

# 在箱线图中绘制出交叉验证结果
# 对分类器性能进行可视化比较
>>> search_results = pd.DataFrame.from_dict({
...                               'bow': bow_search.cv_results_['mean_test_score'],
...                               'tfidf': tfidf_search.cv_results_['mean_test_score'],
...                               'l2': l2_search.cv_results_['mean_test_score']
...                      })

# 常用的matplotlib设置
# seaborn用来美化图形
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> sns.set_style("whitegrid")
```

```
>>> ax = sns.boxplot(data=search_results, width=0.4)
>>> ax.set_ylabel('Accuracy', size=14)
>>> ax.tick_params(labelsize=14)
```

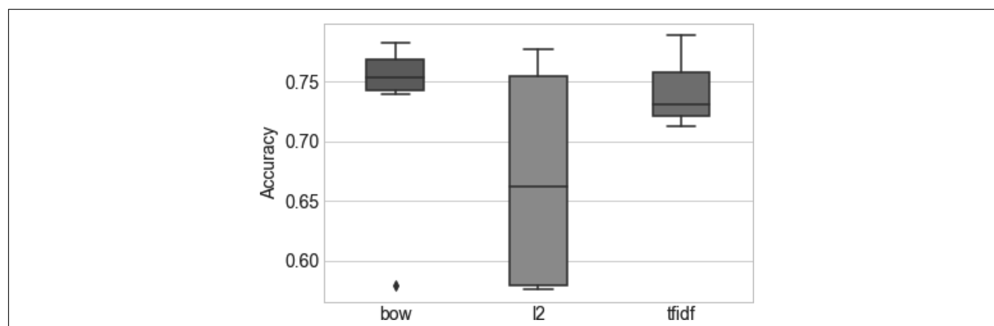


图 4-4：每种特征集合和正则化设置下的分类器准确度分布——使用 5-折交叉验证的平均值作为准确度测量结果

表 4-2 给出了每种超参数设置下的交叉验证分类器准确率的平均值。每列中带星号的值表示该特征集合能达到的最高准确率。

表4-2：交叉验证分类器准确率的平均得分

正则化参数	词　　袋	ℓ^2归一化	tf-idf
0.00001	0.578971	0.575724	0.721638
0.001	0.751811	0.575724	0.788648 *
0.1	0.782839 *	0.589120	0.763566
1	0.773818	0.734247	0.741150
10	0.755160	0.776756 *	0.721467
100	0.739373	0.761106	0.712309

在图 4-4 中，可以看出 ℓ^2 归一化特征的结果非常糟糕。但不要被这张图蒙蔽，准确率如此之低是因为使用了非常不合适的正则化参数设置。这个具体的例子说明了不合适的超参数设置可以导致得出非常错误的结论。如果使用每种特征集合的最优超参数设置来训练模型，那么不同特征集合的准确率得分是非常接近的，如例 4-6 所示。

例 4-6　比较不同特征集合的最终训练与测试步骤

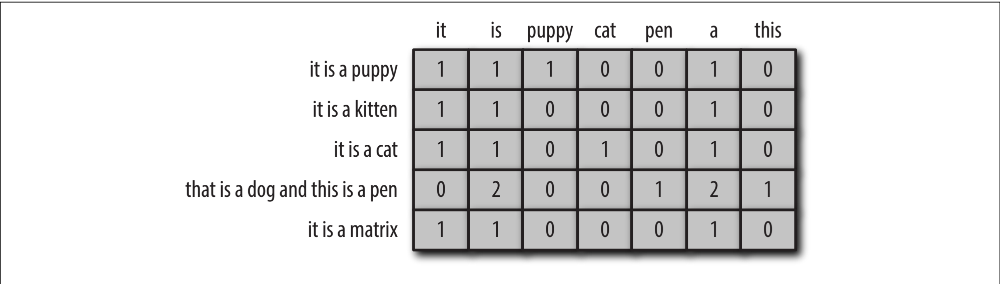
```
# 使用前面找到的最优超参数设置，在整个训练集上训练一个最终模型
# 在测试集上测量准确度
>>> m1 = simple_logistic_classify(X_tr_bow, y_tr, X_te_bow, y_te, 'bow',
...                               _C=bow_search.best_params_['C'])
>>> m2 = simple_logistic_classify(X_tr_l2, y_tr, X_te_l2, y_te, 'l2-normalized',
...                               _C=l2_search.best_params_['C'])
>>> m3 = simple_logistic_classify(X_tr_tfidf, y_tr, X_te_tfidf, y_te, 'tf-idf',
...                               _C=tfidf_search.best_params_['C'])
Test score with bow features: 0.78360708021
Test score with l2-normalized features: 0.780178599904
Test score with tf-idf features: 0.788470738319
```

恰当地调优可以提高所有特征集合的准确率，这三种特征集合经过正则化逻辑回归后，都得到了相似的分类准确率。tf-idf 模型的准确率稍稍高一些，但这种差别似乎不是统计显著的。这些结果令我们非常困惑。如果特征缩放的效果并不比普通的词袋表示好，那它到底有什么意义？如果 tf-idf 没有什么意义，那我们为什么还要这么大动干戈？在下一节中，我们将试图回答这些问题。

4.3 深入研究：发生了什么

为了弄清楚结果背后的原因，我们必须知道模型是如何使用特征的。对于逻辑回归这种线性模型，这个过程是通过一个称为**数据矩阵**的中间对象实现的。

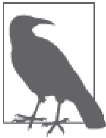
数据矩阵中包含有数据点，它们是由固定长度的扁平向量表示的。如果使用词袋向量，数据矩阵又可以称为**文档－词矩阵**。图 3-1 展示了一个向量形式的词袋向量，图 4-1 演示了特征空间中的 4 个词袋向量。要生成一个文档－词矩阵，只需得到文档向量，把它们放平，再彼此叠加起来即可。这种矩阵的列表示词汇表中所有可能出现的单词（见图 4-5）。因为多数文档只包含所有可能出现的单词中的一小部分，所有矩阵中的多数元素都是 0，这是个**稀疏**矩阵。



	it	is	puppy	cat	pen	a	this
it is a puppy	1	1	1	0	0	1	0
it is a kitten	1	1	0	0	0	1	0
it is a cat	1	1	0	1	0	1	0
that is a dog and this is a pen	0	2	0	0	1	2	1
it is a matrix	1	1	0	0	0	1	0

图 4-5：5 个文档 7 个单词的文档－词矩阵示例

特征缩放实质上是数据矩阵上的列操作。特别地，tf-idf 和 ℓ^2 归一化都是对整个列（例如，一个 n 元词特征）乘以一个常数。



tf-idf = 列缩放

tf-idf 和 ℓ^2 归一化都是数据矩阵上的列操作。

正如附录 A 中所介绍的，训练一个线性分类器其实就是找到特征（数据矩阵的列向量）的最优线性组合。解空间可以由数据矩阵的列空间和零空间表示出来，我们训练出的线性分类器的质量直接依赖于数据矩阵的零空间和列空间。一个巨大的列空间意味着特征之间几乎没有线性相关性，这通常是好事。零空间中包含不能表示为现有数据的线性组合的“奇

异”数据点。一个巨大的零空间会是非常麻烦的。（对于想复习线性决策面、特征分解和矩阵基本子空间这些概念的读者，强烈建议你们仔细阅读一下附录 A。）

列缩放操作对数据矩阵的列空间和零空间有多大的影响？答案是“影响并不大”。但对于 tf-idf 和 ℓ^2 归一化来说，情况会有点不一样。下面解释一下。

由于若干原因，数据矩阵的零空间可以非常大。首先，很多数据集中的数据点彼此非常相似，这意味着与数据集中数据点的数量相比，有效的行空间非常小。其次，特征数量可以比数据点数量多得多。词袋表示法尤其可能创建出巨大的特征空间。在我们的 Yelp 示例中，训练集中有 29 000 条点评和 47 000 个特征。而且，唯一单词的数量通常会随着数据集中文档数量的增加而增加，所以，添加更多文档不一定会降低特征和数据的比例，也不一定会缩减零空间。

通过词袋表示，与特征数量相比，列空间相对较小。可能有些单词在同一文档中出现了大致相同的次数，这会导致相应的列向量几乎是线性相关的，进而导致列空间不是满秩的（尽管它可以满秩）。（满秩的定义请参见附录 A。）这种情况称为**秩亏**。（与动物缺少维生素和矿物质的情况非常类似，矩阵也可以缺少秩，这样输出空间就不像它本应的那么饱满。）

秩亏的行空间和列空间会使模型对问题用力过猛。线性模型会为数据集中的每个特征都配备一个权重参数。如果行空间和列空间是满秩的[1]，模型会生成输出空间中的任何目标向量。当它们秩亏时，模型会具有更多不必要的自由度，这会导致更加难以确定最终解。

特征缩放可以解决数据矩阵的秩亏问题吗？让我们来看一下。

列空间定义为所有列向量的线性组合（加粗斜体字母表示向量）：$a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \cdots + a_n\boldsymbol{v}_n$。特征缩放使用向量与一个常数的乘积来代替这个向量，如：$\boldsymbol{v}_1 = c\boldsymbol{v}_1$。但只要使用 $\tilde{a}_1 = a_1/c$ 代替 a_1，我们还可以生成原来的线性组合，似乎特征缩放不会改变列空间的秩。同样，特征缩放也不会影响零空间的秩，因为我们可以对权重向量中相应的元素进行逆缩放来抵消缩放后的特征列。

但是，这里仍然有一个问题。如果相乘的标量为 0，那么就无法恢复初始的线性组合，$\boldsymbol{v}_1$ 就消失了。如果这个向量与其他列线性无关，那么我们就有效地缩减了列空间并扩展了零空间。

如果这个向量与目标输出不相关，那么我们就有效地消除了噪声信号，这是非常好的事情。这就是 tf-idf 与 ℓ^2 归一化之间的关键区别。ℓ^2 归一化永远不会计算出一个值为 0 的范数，除非向量中都是 0。如果向量接近于 0，那么它的范数也接近于 0。除以一个小范数会突出这个向量，并使它更长。

注 1：严格地说，对于一个长方形矩阵，行空间和列空间不能同时满秩。这两个子空间的最大秩是 m（行数）和 n（列数）中较小的那一个。

另一方面，tf-idf 会生成一个接近于 0 的缩放因子，如图 4-2 所示。当单词出现在训练集中的大量文档中时，会出现这种情况，这种单词很可能与目标向量没有很强的相关性。除去这种单词，可以使解决方案更关注列空间中的其他方向，并找到更好的解（然而准确率的提高幅度很可能不会很大，因为使用这种方法通常找不到太多能削减的噪声方向）。

特征缩放（包括 ℓ^2 归一化和 tf-idf）的真正用武之地是加快解的收敛速度。这表现在它能使数据矩阵具有明显更少的条件数（最大奇异值和最小奇异值的比值，参见附录 A 中关于这些名词的详细讨论）。实际上，ℓ^2 归一化使得条件数几乎为 1。但并不是条件数越少，解就越好。在这次实验中，ℓ^2 归一化收敛得比词袋和 tf-idf 都快得多，但它对过拟合更加敏感：它需要更多的正则化，而且对优化过程中的迭代次数更加敏感。

4.4　小结

在这一章，我们将 tf-idf 作为切入点，详细分析了特征变换对模型的影响。tf-idf 是特征缩放的一个特例，所以我们将它与另一种特征缩放方法——ℓ^2 归一化——的效果进行了对比。

结果并不尽如人意。与普通的词袋表示相比，tf-idf 和 ℓ^2 归一化并没有提高最终分类器的准确率。经过一些统计建模和线性代数分析，我们意识到了原因：它们都没有改变数据矩阵的列空间。

二者之间有个小区别，那就是 tf-idf 既可以“拉长”单词计数，也可以“压缩”它。换句话说，它可以使某些计数变大，同时使其他计数接近于 0。因此，tf-idf 可以比较彻底地消除那些没有信息量的单词。

通过这种方法，我们还发现了特征缩放的另一个作用：它可以减少数据矩阵的条件数，大大加快训练线性模型的速度。ℓ^2 归一化和 tf-idf 都有这种效果。

总而言之，我们的收获是：**正确**的特征缩放有助于分类问题。正确缩放可以突出有信息量的单词，并削弱普通单词的影响。它还可以减少数据矩阵的条件数。正确的缩放不一定是标准的列缩放。

本章内容很好地说明了在一般情况下分析特征工程效果的难度。改变特征会影响训练过程以及随后的模型。尽管线性模型是最容易理解的模型，但还是需要极为细致的实验方法和大量高深的数学知识，才能梳理出理论上和实际的影响。对于更加复杂的模型或特征变换，这种分析几乎是不可能实现的。

4.5　参考文献

Zheng, Alice. Evaluating Machine Learning Models [M]. Sebastopol, CA: O’Reilly Media, 2015.

第 5 章

分类变量：自动化时代的数据计数

顾名思义，**分类变量**是用来表示类别或标记的。例如，分类变量可以表示世界上的主要城市、一年中的四季、企业所属行业（石油、旅游、科技），等等。在实际的数据集中，类别的数量总是有限的。类别可以用数字表示，但与数值型变量不同，分类变量的值是不能被排序的。（作为行业类型，石油和旅游之间是分不出大小的。）它们又称为**无序变量**。

可以用一个简单的问题作为能否使用分类变量的试金石："我们是需要知道两个值有**多大**不同，还是只需要知道它们**是否**不同？"500 美元的股票价格是 100 美元的股票价格的 5 倍，所以，股票价格应该用连续型数值变量表示。另一方面，公司所属行业（石油、旅游、科技，等等）就应该用分类变量表示。

大型分类变量在交易记录中是极其常见的。例如，很多 Web 服务使用 ID 来跟踪用户，ID 就是一个分类变量，它的值根据服务的用户数量的不同，可能有几百个到几亿个。大型分类变量的另一个例子是互联网交易中的 IP 地址。尽管用户 ID 和 IP 地址是用数值表示的，但它们是分类变量，因为它们的大小与当前任务通常是没有关系的。举例来说，当进行个人交易中的欺诈检测时，IP 地址是个相关变量——有些 IP 地址或子网会生成更多的欺诈交易。但地址为 164.203.x.x 的子网不会天生比 164.202.x.x 的子网更具欺诈性，子网地址的数值在这里并不重要。

文档语料库的词汇表可以表示为一个大型分类变量，类别就是唯一的单词。表示如此多的不同类别需要很高的计算成本。如果一个类别（如一个单词）在一个数据点（文档）中出现了多次，就可以将它表示为一个计数，并通过计数统计表示所有类别。这种方法称为**分箱计数**。本章先讨论分类变量的一般表示，然后逐渐过渡到用于大型分类变量的分箱计数方法。在现代数据集中，大型分类变量非常常见。

5.1　分类变量的编码

分类变量中的类别通常不是数值型的。[1] 例如，眼睛的颜色可以是“黑色”“蓝色”和“褐色”，等等。因此，需要一种编码方法来将非数值型的类别转换为数值。我们很容易想到，可以简单地为 k 个可能类别中的每个类别分配一个整数，如从 1 到 k，但这样做的结果是使类别彼此之间有了顺序，这在分类变量中是不允许的。所以，我们要想个别的方法。

5.1.1　one-hot编码

更好的方法是使用一组比特位，每个比特位表示一种可能的类别。如果变量不能同时属于多个类别，那么这组值中就只有一个比特位是“开”的。这就是 **one-hot 编码**，它可以通过 scikit-learn 中的 `sklearn.preprocessing.OneHotEncoder` 实现。每个比特位表示一个特征，因此，一个可能有 k 个类别的分类变量就可以编码为一个长度为 k 的特征向量。表 5-1 给出了一个例子。

表5-1：表示3个城市的分类变量的one-hot编码

	e_1	e_2	e_3
San Francisco	1	0	0
New York	0	1	0
Seattle	0	0	1

one-hot 编码很容易理解，但它使用的比特位要比实际需要的多一位。如果 k−1 位都是 0，那么最后一位肯定是 1，因为变量必须取 k 个值中的一个。在数学上，可以将这个限制条件表述为“所有位的和必须等于 1”：

$$e_1 + e_2 + \cdots + e_k = 1$$

于是就得到了一个线性相关关系。正如我们在第 4 章中发现的那样，线性相关的特征有一点讨厌，因为它们会使训练出的模型不唯一。特征的不同线性组合可以做出同样的预测，所以我们需要做些额外的努力才能理解某个特征对预测结果的作用。

5.1.2　虚拟编码

one-hot 编码的问题是它允许有 k 个自由度，而变量本身只需要 k−1 个自由度。**虚拟编码** [2] 在进行表示时只使用 k−1 个特征，除去了额外的自由度（见表 5-2）。没有被使用的那个特

注 1：在标准的统计学语言中，表示类别的技术术语是**水平**。有两个不同类别的分类变量具有两个水平。但在统计学中，还有一些其他的概念也称为水平，所以本书不使用这个名词，而是使用更加简单明了的名词“类别”。

注 2：好奇的读者会想知道，为什么 one-hot 编码（one-hot encoding）被称为“encoding”，而虚拟编码（dummy coding）被称为“coding”。这主要是历史原因。我猜想 one-hot 编码首先流行于电子工程领域，其中信息总是被编码（encode）和解码（decode）。而另一方面，虚拟编码和效果编码（effect coding）是由统计学社区发明的，由于某种原因，“en”没有在这个学术分支流行起来。

征通过一个全零向量来表示，它称为**参照类**。虚拟编码和 one-hot 编码都可以通过 Pandas 包中的 `pandas.get_dummies` 来实现。

表5-2：表示3个城市的分类变量的虚拟编码

	e_1	e_2
San Francisco	1	0
New York	0	1
Seattle	0	0

使用虚拟编码的模型结果比使用 one-hot 编码的模型结果更具解释性，在简单的线性回归问题中很容易看出这一点。假设有一些公寓租金的数据，这些公寓位于 3 个城市：旧金山（SF）、纽约（NYC）和西雅图（Seattle），见表 5-3。

表5-3：3个城市公寓价格的模拟数据集

	城　市	租　金
0	SF	3999
1	SF	4000
2	SF	4001
3	NYC	3499
4	NYC	3500
5	NYC	3501
6	Seattle	2499
7	Seattle	2500
8	Seattle	2501

可以仅基于城市标识训练一个线性回归器来预测租金（见例 5-1）。

线性回归模型可以写成如下形式：

$$y = w_1x_1 + \cdots + w_nx_n$$

通常，我们还要加上一个额外的常数，称为**截距**，使得当 x 的值都为 0 时 y 值可以不为 0：

$$y = w_1x_1 + \cdots + w_nx_n + b$$

例 5-1　对使用 one-hot 编码和虚拟编码的分类变量进行线性回归

```
>>> import pandas
>>> from sklearn import linear_model

# 定义一个模拟数据集，表示纽约、旧金山和西雅图的公寓租金
>>> df = pd.DataFrame({
...     'City': ['SF', 'SF', 'SF', 'NYC', 'NYC', 'NYC',
...              'Seattle', 'Seattle', 'Seattle'],
...     'Rent': [3999, 4000, 4001, 3499, 3500, 3501, 2499, 2500, 2501]
... })
```

```
>>> df['Rent'].mean()
3333.3333333333335

# 将数据框中的分类变量转换为one-hot编码并拟合一个线性回归模型
>>> one_hot_df = pd.get_dummies(df, prefix=['city'])
>>> one_hot_df
   Rent  city_NYC  city_SF  city_Seattle
0  3999    0.0       1.0       0.0
1  4000    0.0       1.0       0.0
2  4001    0.0       1.0       0.0
3  3499    1.0       0.0       0.0
4  3500    1.0       0.0       0.0
5  3501    1.0       0.0       0.0
6  2499    0.0       0.0       1.0
7  2500    0.0       0.0       1.0
8  2501    0.0       0.0       1.0

>>> model = linear_regression.LinearRegression()
>>> model.fit(one_hot_df[['city_NYC', 'city_SF', 'city_Seattle']],
...           one_hot_df['Rent'])
>>> model.coef_
array([ 166.66666667,  666.66666667, -833.33333333])
>>> model.intercept_
3333.3333333333335

# 为虚拟编码训练一个线性回归模型，指定drop_first标志来生成虚拟编码
>>> dummy_df = pd.get_dummies(df, prefix=['city'], drop_first=True)
>>> dummy_df
   Rent  city_SF  city_Seattle
0  3999    1.0       0.0
1  4000    1.0       0.0
2  4001    1.0       0.0
3  3499    0.0       0.0
4  3500    0.0       0.0
5  3501    0.0       0.0
6  2499    0.0       1.0
7  2500    0.0       1.0
8  2501    0.0       1.0

>>> model.fit(dummy_df[['city_SF', 'city_Seattle']], dummy_df['Rent'])
>>> model.coef_
array([  500., -1000.])
>>> model.intercept_
3500.0
```

使用 one-hot 编码时，截距项表示目标变量 `Rent` 的整体均值，每个线性系数表示相应城市的租金均值与整体均值有多大差别。

使用虚拟编码时，偏差系数表示响应变量 y 对于参照类的均值，本例中参照类是纽约。第 i 个特征的系数等于第 i 个类别的均值与参照类均值的差。

在表 5-4 中，你可以清楚地看出这两种编码方法会生成相差很大的线性模型系数。

表5-4：线性回归系数

	x_1	x_2	x_3	b
one-hot 编码	166.67	666.67	−833.33	3333.33
虚拟编码	0	500	−1000	3500

5.1.3　效果编码

另一种分类变量编码是**效果编码**。效果编码与虚拟编码非常相似，区别在于参照类是用全部由 −1 组成的向量表示的，参见表 5-5。

表5-5：表示3个城市的分类变量的效果编码

	e_1	e_2
San Francisco	1	0
New York	0	1
Seattle	−1	−1

效果编码与虚拟编码非常相似，但它的线性回归模型更容易解释。例 5-2 演示了使用效果编码作为输入的情况。截距项表示目标变量的整体均值，各个系数表示了各个类别的均值与整体均值之间的差。（这称为类别或水平的**主效果**，效果编码的名称就是由此而来。）one-hot 编码实际上也可以得到同样的截距和系数，但它的每个城市都有一个线性系数。在效果编码中，没有单独的特征来表示参照类，所以参照类的效果需要单独计算，它是所有其他类别的系数的相反数之和。（参见 UCLA IDRE 网站上的"FAQ: What is effect coding?"以获得更多详细信息。）

例 5-2　使用效果编码的线性回归

```
>>> effect_df = dummy_df.copy()
>>> effect_df.ix[3:5, ['city_SF', 'city_Seattle']] = -1.0
>>> effect_df
   Rent  city_SF  city_Seattle
0  3999    1.0       0.0
1  4000    1.0       0.0
2  4001    1.0       0.0
3  3499   -1.0      -1.0
4  3500   -1.0      -1.0
5  3501   -1.0      -1.0
6  2499    0.0       1.0
7  2500    0.0       1.0
8  2501    0.0       1.0

>>> model.fit(effect_df[['city_SF', 'city_Seattle']], effect_df['Rent'])
>>> model.coef_
array([ 666.66666667, -833.33333333])
>>> model.intercept_
3333.3333333333335
```

5.1.4 各种分类变量编码的优缺点

one-hot 编码、虚拟编码和效果编码彼此之间非常相似，它们都有各自的优缺点。one-hot 编码有冗余，这会使得同一个问题有多个有效模型，这种非唯一性有时候比较难以解释。它的优点是每个特征都明确对应一个类别，而且可以把缺失数据编码为全零向量，模型输出也是目标变量的总体均值。

虚拟编码和效果编码没有冗余，它们可以生成唯一的可解释的模型。虚拟编码的缺点是不太容易处理缺失数据，因为全零向量已经映射为参照类了。它还会将每个类别的效果表示为与参照类的相对值，这看上去有点不直观。

效果编码使用另外一种编码表示参照类，从而避免了这个问题，但是全由 −1 组成的向量是个密集向量，计算和存储的成本都比较高。正是因为这个原因，像 Pandas 和 scikit-learn 这样的常用机器学习软件包更喜欢使用虚拟编码或 one-hot 编码，而不是效果编码。

当类别的数量变得非常大时，这 3 种编码方式都会出现问题，所以需要另外的策略来处理超大型分类变量。

5.2 处理大型分类变量

互联网上的自动数据采集可以生成大型分类变量，在定向广告和欺诈检测这样的应用中，这种情况非常常见。

在定向广告应用中，我们的任务是为一个用户匹配一组广告。这时的特征包括用户 ID、广告的站点域名、查询语句、当前页以及这些特征的所有成对组合。（查询语句是一个文本字符串，可以被分解转换成一般的文本特征。但是，查询语句一般很短，而且通常由短语组成，所以这时最好的做法是保持它们原封不动或者通过一个散列函数来传递，以使得存储和比较更加容易。稍后会详细地讨论散列操作。）这些特征中的每一个都是非常巨大的分类变量，我们面临的挑战就是找到一种合适的特征表示方法，既要内存高效，又能生成精确的、训练速度很快的模型。

现有的解决方案可以分类（哈哈）如下。

(1) 不在编码问题上搞什么花样，使用一个简单、容易训练的模型，在很多机器上使用 one-hot 编码训练线性模型（逻辑回归或线性支持向量机）。

(2) 压缩特征，有两种方式。

  a) 特征散列化，通常用于线性模型。
  b) 分箱计数，常用于线性模型和树模型。

使用一般的 one-hot 编码也是个不错的选择。对于微软公司的广告搜索引擎，Graepel 等人

就在一个贝叶斯 probit 回归模型（可用简单的更新在线训练）中使用了这种二值特征。与此同时，其他一些研究小组则致力于特征压缩方法。Yahoo! 的研究者们对特征散列化推崇备至（Weinberger 等，2009），但是 McMahan 等人在 Google 广告引擎上试验了特征散列化，却没有取得什么显著的进展。而微软的其他研究者又在实践分箱计数这种思想了（Bilenko, 2015）。

正如我们将看到的，所有方法都有各自的优点和缺点。我们先介绍方法本身，然后再讨论它们的利弊得失。

5.2.1　特征散列化

散列函数是一种确定性函数，它可以将一个可能无界的整数映射到一个有限的整数范围 [1, m] 中。因为输入域可能大于输出范围，所以可能有多个值被映射为同样的输出，这称为**碰撞**。**均匀散列函数**可以确保将大致相同数量的数值映射到 m 个分箱中。

我们可以形象地将散列函数想象为一台机器，它吸入一些带数字标号的圆球（键），再把它们分发到 m 个分箱中。标有同样数字的球总是被分发到同一个分箱中（见图 5-1）。散列函数在保持特征空间的同时，又可以在机器学习的训练和评价周期中减少存储空间和处理时间。

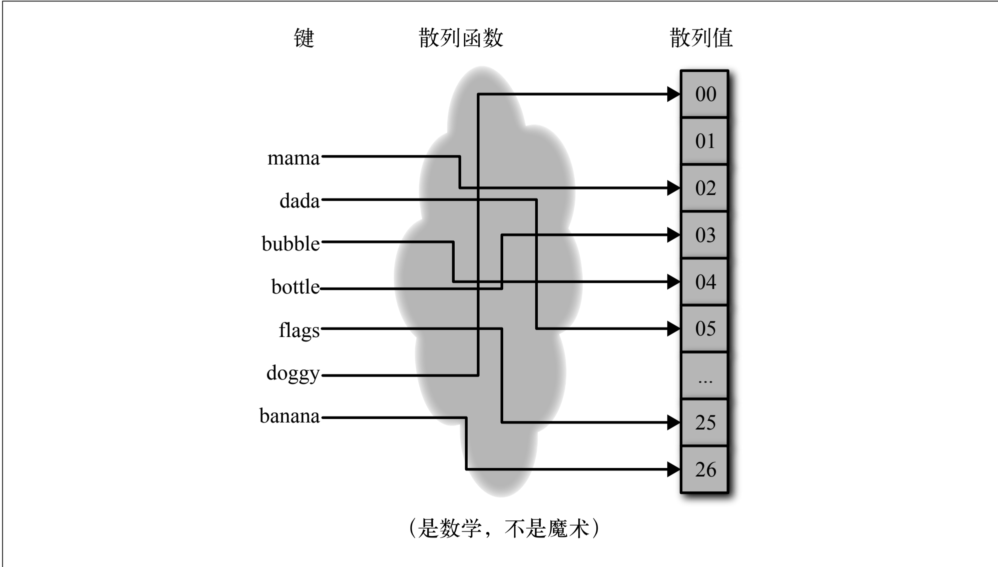


图 5-1：散列函数可以将键映射到分箱

我们可以为任何能表示为数值的对象（也就是任何能存储在计算机上的数据）构造散列函数，这些对象包括数值、字符串、复杂结构，等等。

当特征很多时，保存特征向量需要大量空间。通过对特征 ID 应用散列函数，特征散列化可以将初始特征向量压缩为 m 维向量，如例 5-3 所示。举例来说，如果初始特征是一篇文档中的单词，那么不管输入散列函数的有多少唯一单词，散列化之后的特征都应该是固定长度为 m 的词汇表。

例 5-3　单词特征的特征散列化

```
>>> def hash_features(word_list, m):
...     output = [0] * m
...     for word in word_list:
...         index = hash_fcn(word) % m
...         output[index] += 1
...     return output
```

特征散列化的另一种变体是添加一个正负号，这样就可以从散列后的分箱中加上或减去计数（见例 5-4）。用统计学术语说，这样可以确保散列后特征之间的内积等于初始特征内积的期望。

例 5-4　带符号的特征散列化

```
>>> def hash_features(word_list, m):
...     output = [0] * m
...     for word in word_list:
...         index = hash_fcn(word) % m
...         sign_bit = sign_hash(word) % 2
...         if (sign_bit == 0):
...             output[index] -= 1
...         else:
...             output[index] += 1
...     return output
```

散列后的内积值是初始内积的 $O\left(\frac{1}{\sqrt{m}}\right)$，所以可以根据能接受的误差来选择散列表的长度 m。实际上，可以使用试错法来选择正确的 m。

如果模型中涉及特征向量和系数的内积运算，那么就可以使用特征散列化，比如线性模型和核方法。在垃圾邮件过滤任务中，这种方法已经取得了成功（Weinberger 等，2009）。在定向广告应用中，根据 McMahan 等人的报告（2013），除非 m 达到 10 亿的级别，否则不能将预测误差降低到可接受的水平，在这种情况下，不会有足够的空间节省。

特征散列化的一个缺点是散列后的特征失去了可解释性，只是初始特征的某种聚合。

在例 5-5 中，我们使用 scikit-learn 的 `FeatureHasher` 函数，在 Yelp 点评数据集上演示存储空间和解释性的这种取舍。

例 5-5　特征散列化（又称“散列戏法”）

```
>>> import pandas as pd
>>> import json
```

```
# 加载前10 000条点评
>>> f = open('yelp_academic_dataset_review.json')
>>> js = []
>>> for i in range(10000):
...     js.append(json.loads(f.readline()))
>>> f.close()
>>> review_df = pd.DataFrame(js)

# 定义m为唯一的business_id的数量
>>> m = len(review_df.business_id.unique())
>>> m
528

>>> from sklearn.feature_extraction import FeatureHasher
>>> h = FeatureHasher(n_features=m, input_type='string')
>>> f = h.transform(review_df['business_id'])

# 散列化对特征可解释性的影响
>>> review_df['business_id'].unique().tolist()[0:5]
['vcNAWiLM4dR7D2nwwJ7nCA',
 'UsFtqoBl7naz8AVUBZMjQQ',
 'cE27W9VPgO88Qxe4ol6y_g',
 'HZdLhv6COCleJMo7nPl-RA',
 'mVHrayjG3uZ_RLHkLj-AMg']

>>> f.toarray()
array([[ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ...,  0.,  0.,  0.]])

# 不是很好，但是看一下特征的存储空间大小
>>> from sys import getsizeof
>>> print('Our pandas Series, in bytes: ', getsizeof(review_df['business_id']))
>>> print('Our hashed numpy array, in bytes: ', getsizeof(f))
Our pandas Series, in bytes:  790104
Our hashed numpy array, in bytes:  56
```

我们可以清楚地看到，特征散列化对计算能力大有裨益，但牺牲了直观的用户可解释性。对于大数据集，当从数据探索和可视化进展到机器学习流程时，我们可以很容易地在二者之间做出取舍。

5.2.2 分箱计数

分箱计数是机器学习中的重新发现之一，从广告点击率预测到硬件分支预测，很多应用都对它进行了改造并使用（Yeh and Patt, 1991；Lee 等，1998；Chen 等，2009；Li 等，2010）。但因为它是一种特征工程技术，不是建模或优化方法，所以没有关于它的研究论文。关于

这种技术最详细的描述可以在 Misha Bilenko 的博客文章“Big Learning Made Easy—with Counts!”及其相关幻灯片中找到。

分箱计数的思想稍有一点复杂：它不使用分类变量的**值**作为特征，而是使用目标变量取这个值的**条件概率**。换句话说，我们不对分类变量的值进行编码，而是要计算分类变量值与要预测的目标变量之间的相关统计量。对于那些熟悉朴素贝叶斯分类器的人来说，这个统计量肯定耳熟能详，因为它就是在所有特征都是独立的这个假设之下的各个类别的条件概率。最好通过一个例子来说明它（见表 5-6）。

表5-6：分箱计数特征示例（根据“Big Learning Made Easy—with Counts!”一文进行重制，已获得许可）

用户	点击数	未点击数	点击概率	查询散列值、广告域	点击数	未点击数	点击概率
Alice	5	120	0.0400	0x598fd4fe, foo.com	5000	30000	0.167
Bob	20	230	0.0800	0x50fa3cc0, bar.org	100	900	0.100
…				…			…
Joe	3	3	0.400	0x437a45e1, qux.net	6	18	0.250

分箱计数假设可以使用历史数据来计算统计量。表 5-6 包含了分类变量的每个可能取值的累积历史计数。根据用户 Alice 点击广告的次数和她没有点击广告的次数，可以计算出她点击广告的概率。同样，也可以计算出任意一个查询 – 广告域组合的点击概率。在训练模型时，只要遇到 Alice，就可以使用她的**点击概率**作为模型的输入特征。对于像“0x437a45e1, qux.net”（查询散列值 – 广告域）这样的成对特征，也可以进行同样的处理。

假设有 10 000 个用户，one-hot 编码会生成一个长度为 10 000 的稀疏向量，只在对应当前数据点的列上有一个 1。分箱计数会将所有 10 000 个二值列编码为一个单独的特征，是 0 和 1 之间的一个实数值。

除了历史点击概率，我们还可以包括其他特征：原始计数本身（点击数和未点击数）、对数优势比，或任何其他概率衍生指标。本节的例子是预测广告点击率，但现有技术是用来预测一般性的二值分类问题的。通过常用的将二值分类扩展为多值分类的技术，如一对多优势比或其他多类别标记编码技术，我们很容易扩展到多类别分类。

用于分箱计数的优势比和对数优势比

优势比通常定义在两个二值变量之间，它通过这样一个问题来衡量两个变量之间的联系强度：“当 X 为真时，Y 在多大程度上更可能为真？”例如，我们可以这样提问：“Alice 在多大程度上比一般人更可能点击这个广告？”在这里，X 是表示“Alice 是当前用户”的二值变量，Y 是表示“是否点击广告”的变量。优势比的计算要使用双向列联表（表中的 4 个数字分别对应于 X 和 Y 的 4 种可能组合），参见表 5-7。

表5-7：表示广告点击和用户的列联表

	点击	未点击	总计
Alice	5	120	125
不是 Alice	995	18 880	19 875
总计	1000	19 000	20 000

给定输入变量 X 和目标变量 Y，优势比定义如下：

$$\text{优势比} = \frac{P(Y=1 \mid X=1)\,/\,P(Y=0 \mid X=1)}{P(Y=1 \mid X=0)\,/\,P(Y=0 \mid X=0)}$$

在我们的例子中，这个公式可以转换为“Alice 点击广告的概率在多大程度上高于她不点击广告的概率”和“其他人点击广告的概率在多大程度上高于他们不点击广告的概率”这两个问题的比。这样，实际数值为：

$$\text{优势比（用户，广告点击）} = \frac{(5/125)\,/\,(120/125)}{(995/19\,875)\,/\,(18\,880/19\,875)} = 0.17906$$

再简单一些，我们只看分子，它表示一个用户（Alice）点击广告的可能性比不点击广告的可能性大多少，这个定义也适合具有多个值（不仅两个值）的大型分类变量：

$$\text{优势比（用户，广告点击）} = \frac{5/125}{120/125} = 0.04166$$

概率比值很容易特别大或特别小。（例如，会有几乎从来不点击广告的用户，也会有频繁点击广告的用户。）这时我们又需要使用对数变换了。对数运算另一个有用的性质是可以将除法转换为减法：

$$\text{对数优势比（Alice，广告点击）} = \log\left(\frac{5}{125}\right) - \log\left(\frac{120}{125}\right) = -3.178$$

简而言之，分箱计数将一个分类变量转换为与其值相关的统计量，它可以将一个大型的、稀疏的、二值的分类变量表示（如 one-hot 编码生成的结果）转换为一个小巧的、密集的、实数型的数值表示（见图 5-2）。

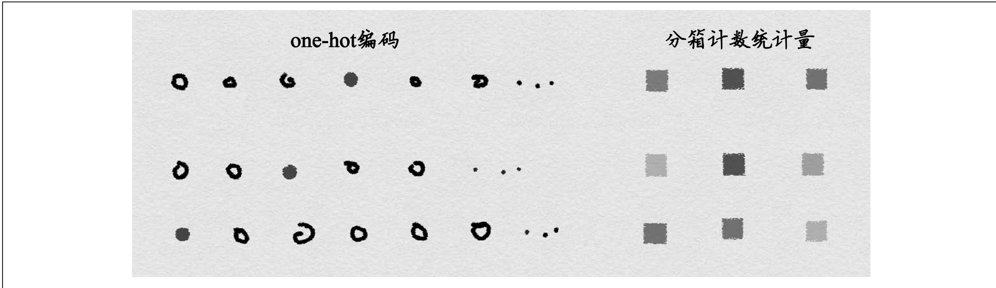


图 5-2：分类变量的 one-hot 编码与分箱计数统计量

至于具体的实现，分箱计数需要保存每个类别与其相应计数之间的映射关系。（统计量的其他计算工作可以根据原始计数直接导出。）因此，它需要 $O(k)$ 大小的空间，其中 k 是分类变量中唯一值的个数。

为了实际演示一下分箱计数，我们要使用一下由 Avazu 主办的 Kaggle 竞赛中的数据。下面是这个数据集的几个相关统计量。

- 共有 24 个变量，其中 `click` 是个二值计数器，值为 click 和 no click，`device_id` 用来跟踪广告显示在哪个设备上。
- 整个数据集包括 40 428 967 条观测，使用了 2 686 408 台独立的设备。

Avazu 竞赛的目标是使用广告数据预测点击率，但我们要使用这个数据集做一个演示，说明对于大量的流式数据，分箱计数是如何显著缩减特征空间的（见例 5-6）。

例 5-6　分箱计数示例

```
>>> import pandas as pd

# 使用这个超过6GB的数据集的前10 000行作为训练集
>>> df = pd.read_csv('data/train_subset.csv')

# 看看训练集中有多少个唯一的特征
>>> len(df['device_id'].unique())
7201

# 对每个类别，我们要计算：
# Theta = [counts, p(click), p(no click), p(click)/p(no click)]

>>> def click_counting(x, bin_column):
...     clicks = pd.Series(x[x['click'] > 0][bin_column].value_counts(),
...                        name='clicks')
...     no_clicks = pd.Series(x[x['click'] < 1][bin_column].value_counts(),
...                           name='no_clicks')

...     counts = pd.DataFrame([clicks,no_clicks]).T.fillna('0')
...     counts['total_clicks'] = counts['clicks'].astype('int64') +
...                              counts['no_clicks'].astype('int64')
...     return counts

>>> def bin_counting(counts):
...     counts['N+'] = counts['clicks']
...                    .astype('int64')
...                    .divide(counts['total_clicks'].astype('int64'))
...     counts['N-'] = counts['no_clicks']
...                    .astype('int64')
...                    .divide(counts['total_clicks'].astype('int64'))
...     counts['log_N+'] = counts['N+'].divide(counts['N-'])
...     # 如果只想返回分箱属性就进行过滤
...     bin_counts = counts.filter(items= ['N+', 'N-', 'log_N+'])
...     return counts, bin_counts
```

```
# 分箱计数示例：device_id
>>> bin_column = 'device_id'
>>> device_clicks = click_counting(df.filter(items=[bin_column, 'click']),
...                               bin_column)
>>> device_all, device_bin_counts = bin_counting(device_clicks)

# 检查一下，确定我们处理了所有设备
>>> len(device_bin_counts)
7201

>>> device_all.sort_values(by = 'total_clicks', ascending=False).head(4)

         clicks  no_clicks  total  N+        N-        log_N+
a99f214a 15729   71206      86935  0.180928  0.819072  0.220894
c357dbff 33      134        167    0.197605  0.802395  0.246269
31da1bd0 0       62         62     0.000000  1.000000  0.000000
936e92fb 5       54         59     0.084746  0.915254  0.092593
```

1. 如何处理稀有类

和罕见词一样，稀有类也需要特殊处理。设想一个一年只登录一次的用户：只有极少的数据能用来可靠地估计这个用户的广告点击率。而且，稀有类还会浪费计数表中的空间。

解决这个问题的一种方法称为 **back-off**，这是一种将所有稀有类的计数累加到一个特殊分箱中的简单技术（见图 5-3）。如果类别的计数大于一个确定的阈值，那么就使用它自己的计数统计量；否则，就使用 back-off 分箱的统计量。这种方法本质上就是把单个稀有类的统计量转换为使用所有稀有类计算的统计量。当使用 back-off 方法时，可以添加一个表示统计量是否来自于 back-off 分箱的二值指示器。

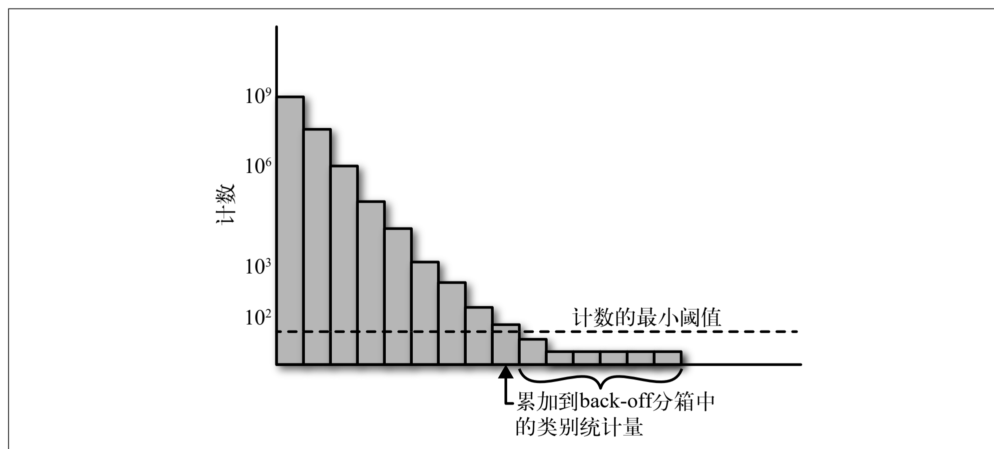


图 5-3：如果一个稀有类的计数超过了 back-off 分箱阈值，就使用它自己的计数统计量进行建模

解决这个问题的另外一种方法称为**最小计数图**（Cormode and Muthukrishnan, 2005）。在这种方法中，不管是稀有类还是频繁类，所有类别都通过多个散列函数进行映射，每个散列

函数的输出范围 m 都远远小于类别数量 k。在计算统计量时，需要使用所有散列函数进行计算，并返回结果中最小的那个统计量。与使用单散列函数相比，使用多个散列函数可以降低碰撞概率。这种方法的有效之处在于，散列函数的数量乘以散列表大小 m 之后，不但可以小于类别数量 k，而且能保持非常低的碰撞概率。

图 5-4 演示了这个过程。对于每个项目 i，都把它映射到计数数组每一行中的某个单元。当项目 i_t 的计数 c_t 更新时，就使用函数 $h_1 \cdots h_d$ 进行散列，添加到每个单元中。

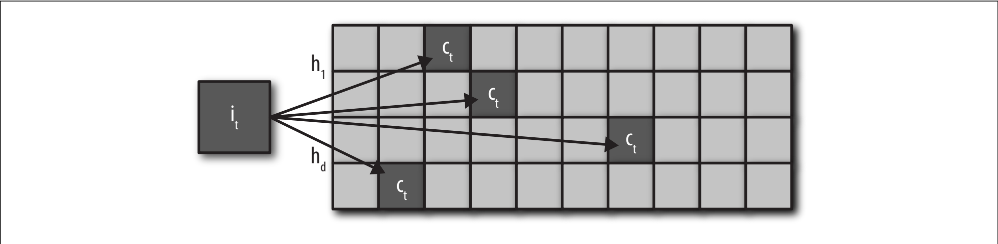


图 5-4：最小计数图

2. 防止数据泄露

因为分箱计数要依赖历史数据生成必需的统计量，所以它需要等待一段时间以完成数据收集，这就会在学习流程中导致一点轻微的延迟。还有，当数据分布改变时，需要更新计数。数据变化得越快，计数重新计算的频率就越高。在像定向广告这样的应用中，用户偏好和常用查询变化得非常快，所以这个问题变得特别重要，不能适应当前数据分布的变化意味着广告平台的巨大损失。

有人或许会问：为什么不使用同样的数据集来计算相关统计量和训练模型？这种想法太天真了。这里最大的问题是，统计量中包含目标变量，而它正是模型试图去预测的。使用输出去计算输入特征会导致一个非常严重的问题，那就是**数据泄露**。简单地说，数据泄露会使模型中包含一些不应该有的信息，这些信息可以使模型获得某种不现实的优势，从而做出更加精确的预测。出现数据泄露有多种原因，比如测试数据泄露到训练数据中，或者未来数据泄露到过去数据中。只要模型获得了在生产环境中实时预测时不应该接触到的信息，就会发生数据泄露。Kaggle 的 wiki 中给出了更多数据泄露的例子，以及它不利于机器学习应用的原因。

如果在分箱计数过程中使用当前数据点的标签来计算输入统计量，就会造成直接的数据泄露。防止出现这个问题的一种方法是，严格隔离计数收集（用来计算分箱计数统计量）和训练，如图 5-5 所示；也就是说，使用过去的数据点进行计数，使用当前的数据点进行训练（将分类变量映射到我们前面收集到的历史统计量上），再使用未来的数据点进行测试。这可以解决数据泄露问题，但会引发前面提过的流程延迟问题（输入统计量以及模型会滞后于当前数据）。

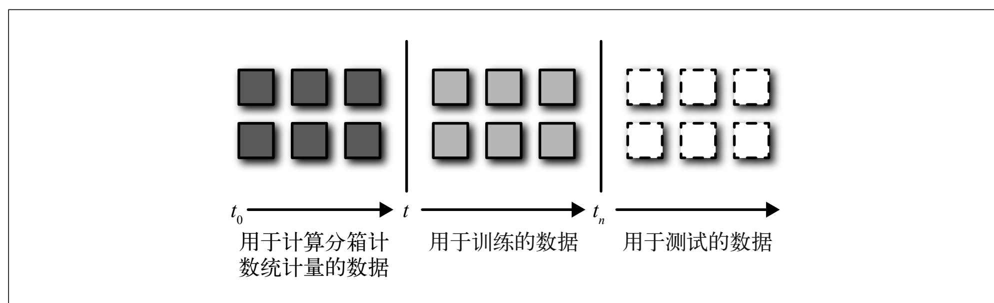


图 5-5：使用时间窗口可以防止分箱计数过程中的数据泄露

还有一种基于差分隐私的解决方案。对于一个统计量，如果不管有没有任何一个数据点，它的分布都保持基本不变，那么它就是**近似防漏**的。实际上，使用 Laplace(0,1) 分布添加一个小的随机噪声，就足以弥补任何来自单数据点的潜在泄露。这种思想可以和留一计数方法结合起来，构成用于当前数据的统计量（Zhang, 2015）。

3. 无界计数

如果提供的历史数据越来越多，统计量持续更新，计数就会无限增长。这对于模型来说是个问题。一个训练好的模型应该“知道”输入数据的可见范围。训练好的决策树可以这样表述：“当 x 大于 3 时，预测值为 1。”训练好的线性模型可以这样表述：“将 x 乘以 0.7，然后看看结果是否大于全局平均数。”当 x 位于 0 和 5 之间时，这些可能是正确的决策。但如果超出这个范围呢？没有人知道。

当输入计数增加时，模型需要维持原来的规模。如果计数累积得比较慢，有效范围不会变得太快，模型就不需要维护得特别频繁。但当计数增加得非常快时，过于频繁的维护就会造成很多问题。

由于这个原因，通常更好的做法是使用归一化后的计数，这样就可以保证把计数值限制在一个可知的区间中。例如，点击率的估计值被限制在 [0, 1] 这一范围。另一种方法是进行对数变换，这样可以强加一个严格的边界，但当计数值非常大时，变换结果的增加速度是非常慢的。

这两种方法都不能保证输入分布保持不变（例如，去年的芭比娃娃已经过时了，人们不会再点击那些广告）。需要对模型进行维护，以适应输入数据分布中这些更加基本的改变，或者将整个流程转换为在线学习模式，使得模型能持续地适应输入变化。

5.3 小结

本章详细介绍的每种方法都有各自的优点和缺点，下面是一个简短的总结。

普通one-hot编码	
空间要求	使用稀疏向量格式时为 $O(n)$，其中 n 是数据点的个数
计算能力要求	线性模型下为 $O(nk)$，其中 k 是类别数量
优点	• 容易实现 • 可能是最精确的 • 可用于在线学习
缺点	• 计算效率不高 • 不能适应可增长的类别 • 只适用于线性模型 • 对于大数据集，需要大规模的分布式优化

特征散列化	
空间要求	使用稀疏向量格式时为 $O(n)$，其中 n 是数据点的个数
计算能力要求	线性模型和核方法下为 $O(nm)$，其中 m 是散列分箱的个数
优点	• 容易实现 • 模型训练成本更低 • 容易适应新类别 • 容易处理稀有类 • 可用于在线学习
缺点	• 只适合线性模型或核方法 • 散列后的特征无法解释 • 精确度难以保证

分箱计数	
空间要求	$O(n+k)$，将每个数据点表示为小而密集的向量，加上为每个类别保存的计数统计量
计算能力要求	线性模型下为 $O(n)$，也适用于非线性模型，比如树
优点	• 训练阶段的计算负担最小 • 可用于基于树的模型 • 比较容易适应新类别 • 可使用 back-off 方法或最小计数图处理稀有类 • 可解释
缺点	• 需要历史数据 • 需要延迟更新，不完全适合在线学习 • 很可能导致数据泄露

正如你看到的，没有一种方法是完美无缺的。应该使用哪种方法取决于具体的模型。线性模型的训练成本低，因此可以使用未经压缩的特征表示，比如 one-hot 编码。另一方面，基于树的模型需要在所有特征中重复搜索以进行正确的分割，因此只能使用规模较小的特征表示，比如分箱计数。特征散列化位于这两个极端之间，但结果的准确率众说纷纭。

5.4 参考文献

Agarwal, Alekh, Oliveier Chapelle, Miroslav Dudík, and John Langford. A Reliable Effective Terascale Linear Learning System [J]. Journal of Machine Learning Research 15 (2015): 1111–1133.

Bilenko, Misha. Big Learning Made Easy—with Counts! [EB/OL]. Cortana Intelligence and Machine Learning Blog, February 17, 2015. https://blogs.technet.microsoft.com/machinelearning/2015/02/17/big-learning-made-easy-with-counts/.

Chen, Ye, Dmitry Pavlov, and John F. Canny. Large-Scale Behavioral Targeting [C]. Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2009): 209–218.

Cormode, Graham, and S. Muthukrishnan. An Improved Data Stream Summary: The Count-Min Sketch and Its Applications [G]. Algorithms 55 (2005): 29–38.

Graepel, Thore, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine [C]. Proceedings of the 27th International Conference on Machine Learning (2010): 13–20.

He, Xinran, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. Practical Lessons from Predicting Clicks on Ads at Facebook [C]. Proceedings of the 8th International Workshop on Data Mining for Online Advertising (2014): 1–9.

Lee, Wenke, Salvatore J. Stolfo, and Kui W. Mok. 1998. Mining Audit Data to Build Intrusion Detection Models [C]. Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (1998): 66–72.

Li, Wei, Xuerui Wang, Ruofei Zhang, Ying Cui, Jianchang Mao, and Rong Jin. Exploitation and Exploration in a Performance Based Contextual Advertising System [C]. Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2010): 27–36.

McMahan, H. Brendan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad Click Prediction: A View from the Trenches [C]. Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2013): 1222–1230.

Weinberger, Kilian, Anirban Dasgupta, Josh Attenberg, John Langford, and Alex Smola. 2009. Feature Hashing for Large Scale Multitask Learning [C]. Proceedings of the 26th International Conference on Machine Learning (2009): 1113–1120.

Yeh, Tse-Yu, and Yale N. Patt. Two-Level Adaptive Training Branch Prediction [C]. Proceedings of the 24th Annual International Symposium on Microarchitecture (1991):51–61.

Zhang, Owen. 2015. Tips for data science competitions [EB/OL]. SlideShare presentation. https://www.slideshare.net/OwenZhang2/tips-for-data-science-competitions.

第 6 章

数据降维：使用PCA挤压数据

通过自动数据收集和特征生成技术，可以快速获取大量特征，但不是所有特征都是有用的。第 3 章和第 4 章讨论了基于频率的过滤技术和特征缩放技术，它们可以作为消除无用特征的手段。下面仔细研究一下使用**主成分分析**（principal component analysis，PCA）来降低特征维度这个问题。

从本章开始，我们要研究基于模型的特征工程技术。在此之前介绍的大部分技术可以在不考虑数据的情况下进行定义。例如，基于频率的过滤可以表述为“除去所有小于 n 的计数”，不用考虑数据本身的更多性质，就可以实现这种技术。

与之不同的是，基于模型的技术需要来自于数据的信息。例如，PCA 要根据数据的主轴进行定义。在前面的章节中，数据、特征和模型之间总是有一个明确的界限，但从本章开始，它们之间的区别会变得越来越模糊。这也正是当前特征学习研究的热点所在。

6.1　直观理解

数据降维就是在保留重要信息的同时消除那些“无信息量的信息”。“无信息量”有多种定义方法，PCA 关注的是线性相关性。在附录 A.2 节中，我们将数据矩阵的列空间描述为所有特征向量的生成空间。如果列空间的秩小于特征总数，那么多数特征就是几个关键特征的线性组合。线性相关的特征是对空间和计算能力的浪费，因为它们包含的信息可以从更少的几个特征中推导出来。为了避免这种情况，PCA 试图将数据挤压到一个维度大大小于原空间的线性子空间，从而消除这些“臃肿”。

我们在图中画出特征空间中的数据点集合。每个圆点表示一个数据点，整个数据点集合形

成了一个点团。在图 6-1a 中，数据点均匀地分布在两个特征维度上，点团充满了特征空间。在这个例子中，列空间是满秩的。然而，如果有些特征是其他特征的线性组合，那么点团就不会这么饱满，而是类似于图 6-1b。图中是一个扁平的点团，其中特征 1 是特征 2 的一个复制品（或是一个和标量的乘积）。在这种情况下，我们称点团的**本征维数**为 1，即使它存在于一个二维特征空间中。

实际上，彼此完全相同的特征是极其罕见的，更可能的情况是非常接近于相同但又不完全相同的特征。在这种情况下，表示数据的点团类似于图 6-1c，是一个狭长的点团。如果想削减传递给模型的特征数量，可以用一个新的特征来代替特征 1 和特征 2，不妨称之为特征 1.5，它位于两个初始特征之间的对角线上。这样，初始数据集就可以用一个数值来表示——在特征 1.5 方向上的位置，而不是用两个数值——*f*1 和 *f*2。

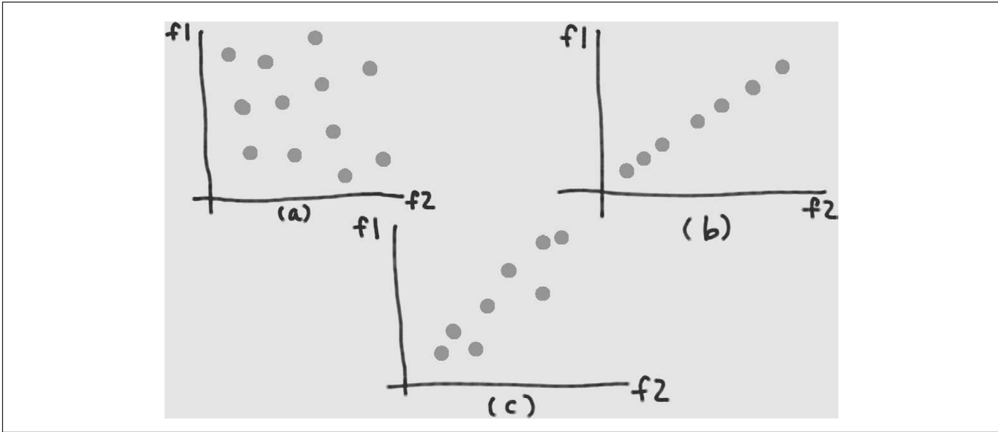


图 6-1：特征空间中的点团：(a) 满秩点团，(b) 低维度点团，(c) 近似低维度点团

PCA 的核心思想是，**使用一些新特征代替冗余特征，这些新特征能恰当地总结初始特征空间中包含的信息**。当只有两个特征时，很容易找出新特征，但当初始特征空间中有成百上千个维度时，就非常困难了。我们需要一种方法，先用数学语言描述要找出的新特征，然后使用最优化技术来找出它们。

对“恰当地总结信息”的一种数学定义是，新的数据点团应该尽量多地保持原数据集中的信息。我们将数据点团挤压成了一个扁平的饼，但我们希望这个饼在正确的方向上尽可能地大。这意味着我们需要一种测量信息量的方式。

信息量肯定与距离有关，但数据集合中的距离表示总是有点模糊。有人使用集合中任意两点之间距离的最大值，但事实证明这种函数很难进行数学优化。另一种方式是使用两点之间距离的平均值，或另一种等价形式——每个数据点与均值之间的平均距离，也就是方差；事实证明这种方式更容易进行优化。（生活本已艰难，幸好统计学家已经找到了捷径。）采用数学方式，这个问题就变成了使新特征空间中数据点的方差最大化。



线性代数公式阅读技巧

要想理解线性代数公式，必须搞清楚哪些量是标量，哪些量是向量，以及向量的方向——垂直的还是水平的。要知道矩阵的维度，因为它们经常会告诉你需要的向量是在行中还是在列中。在一张纸上画出表示矩阵或向量的长方形，确保它们的形状是匹配的。就像注意测量单位（比如用英里表示距离，用英里每小时表示速度）可以让你深刻理解代数问题一样，在线性代数中，你要注意的就是维度。

6.2　数学推导

和前面一样，令 $\boldsymbol{X}$ 表示 $n \times d$ 的数据矩阵，其中 n 是数据点的数量，d 是特征的数量。令 $\boldsymbol{x}$ 是表示单个数据点的列向量（所以 $\boldsymbol{x}$ 是 $\boldsymbol{X}$ 中一行的转置）。令 $\boldsymbol{v}$ 是表示新特征的向量，即要找出的主成分。

矩阵的奇异值分解

任意长方形矩阵都可以分解为 3 个具有特殊形状和性质的矩阵：

$$\boldsymbol{X} = \boldsymbol{U\Sigma V}^{\mathrm{T}}$$

其中 $\boldsymbol{U}$ 和 $\boldsymbol{V}$ 是正交矩阵（即 $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = I$，$\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V} = I$）。$\boldsymbol{\Sigma}$ 是一个包含 $\boldsymbol{X}$ 的奇异值的对角阵，奇异值可以是正数、零或负数。假设 $\boldsymbol{X}$ 有 n 行 d 列，且 $n \geqslant d$，则 $\boldsymbol{U}$ 的形状是 $n \times d$，$\boldsymbol{\Sigma}$ 和 $\boldsymbol{V}$ 的形状是 $d \times d$。（参考附录 A.2.2 节，复习一下矩阵的奇异值分解和特征值分解。）

6.2.1　线性投影

下面我们一步一步地进行 PCA 推导。图 6-2 演示了整个过程。

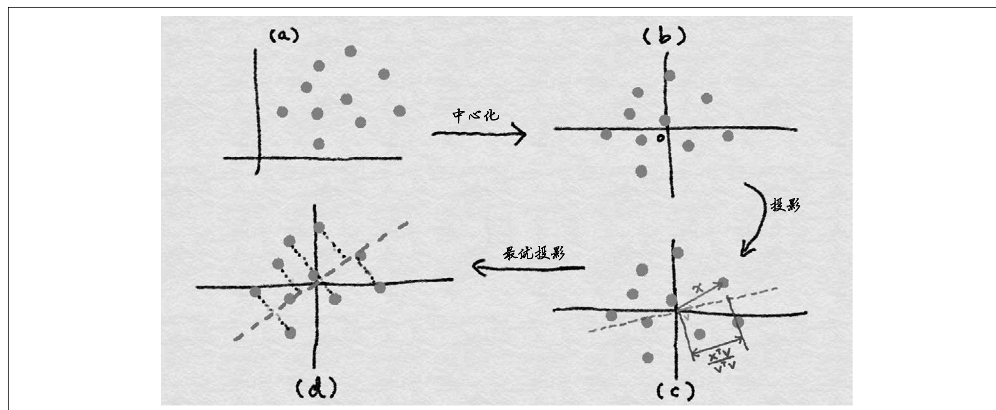


图 6-2：PCA 示意图：(a) 特征空间中的初始数据；(b) 数据中心化；(c) 将数据向量 x 投影到另一个向量 v 上；(d) 投影坐标方差最大的方向（等于 $X^{\mathrm{T}}X$ 的主特征向量）

PCA 使用线性投影将数据转换到新特征空间。图 6-2c 演示了线性投影。当将 $\boldsymbol{x}$ 投影到 $\boldsymbol{v}$ 上时，投影的长度是这两个向量的内积的一个比例，即通过 $\boldsymbol{v}$ 的范数（向量与它本身的内积）进行了归一化。然后，限制 $\boldsymbol{v}$ 具有单位范数。这样，唯一重要的部分就是分子了，我们称其为 $\boldsymbol{v}$（见公式 6-1）。

公式 6-1　投影坐标

$$z = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{v}$$

注意，尽管 $\boldsymbol{x}$ 和 $\boldsymbol{v}$ 是列向量，但 z 是个标量。因为有很多数据点，所以我们可以构造一个向量 $\boldsymbol{z}$，表示所有数据点在新特征 $\boldsymbol{v}$ 上的投影坐标（见公式 6-2）。这里，$\boldsymbol{X}$ 是我们熟悉的数据矩阵，其中每行都是一个数据点。最终结果 $\boldsymbol{z}$ 是个列向量。

公式 6-2　投影坐标向量

$$\boldsymbol{z} = \boldsymbol{X}\boldsymbol{v}$$

6.2.2　方差和经验方差

下一步是计算投影的方差。方差是与均值之间的距离的平方的期望值（见公式 6-3）。

公式 6-3　随机变量 Z 的方差

$$\mathrm{Var}(Z) = \mathrm{E}[Z - \mathrm{E}(Z)]^2$$

还有一个小问题：在我们的问题表示中，从来没有涉及过均值 $\mathrm{E}(Z)$，它是个自由变量。这个问题的一种解决方法是，从所有数据点中减去均值，从而将其从公式中除掉。这样，结果数据集的均值就是 0，这意味着方差就是 Z^2 的期望值。从几何意义上说，减去均值的效果就是将数据中心化（见图 6-2a 和图 6-2b）。

一个与方差关系非常紧密的量是两个随机变量 Z_1 和 Z_2 之间的协方差（见公式 6-4），可以将其视为方差（单随机变量）概念在两个随机变量上的推广。

公式 6-4　两个随机变量 Z_1 和 Z_2 之间的协方差

$$\mathrm{Cov}(Z_1,Z_2) = \mathrm{E}[(Z_1 - \mathrm{E}(Z_1))(Z_2 - \mathrm{E}(Z_2))]$$

当随机变量的均值为 0 时，它们的协方差与**线性相关度** $\mathrm{E}(Z_1Z_2)$ 一致。随后会讨论这个概念。

像方差和期望这样的统计量是定义在数据分布之上的。实际上，我们不可能有真正的数据分布，只有一系列观测到的数据点，$z_1, \cdots, z_n$。这称为**经验分布**，它可以给出方差的经验估计（见公式 6-5）。

公式 6-5　Z 的经验方差，基于观测 z 计算

$$\mathrm{Var}_{\mathrm{emp}}(Z) = \frac{1}{n-1}\sum_{i=1}^{n} z_i^2$$

6.2.3 主成分：第一种表示形式

结合公式 6-1 中 z_i 的定义，可以得到公式 6-6 中投影数据方差最大化的数学表示。（我们丢弃了经验方差定义中的分母 $n–1$，因为它是个全局常量，对在何处达到最大值没有影响。）

公式 6-6 主成分目标函数

$$\max_{\boldsymbol{w}} \sum_{i=1}^{n} (\boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{w})^2, \text{其中 } \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} = 1$$

这里的限制条件强制 $\boldsymbol{w}$ 与自己的内积为 1，这等价于 $\boldsymbol{w}$ 必须具有单位长度。这样做的原因是我们只关心 $\boldsymbol{w}$ 的方向，不关心它的大小。$\boldsymbol{w}$ 的大小是个不必要的自由度，所以我们可以将它设定为任意值，从而除去它的影响。

6.2.4 主成分：矩阵-向量表示形式

下面的步骤有点难度。公式 6-6 中的平方和项太笨重了，表示成矩阵－向量的形式会更简洁。可以这样做吗？答案是肯定的。关键就在于平方和恒等式：多个项的平方和等于由这些项组成的向量的范数的平方，也就是这个向量与它自己的内积。有了这个恒等式，就可以将公式 6-6 重新表示为矩阵－向量形式，如公式 6-7 所示。

公式 6-7 主成分目标函数，矩阵－向量形式

$$\max_{\boldsymbol{w}} \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w}\text{，其中 } \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} = 1$$

这种 PCA 表示形式更加清楚地呈现出了我们的目标：找到一个能使输出向量的范数最大化的输入方向。是不是听起来很熟悉？答案就在于矩阵 $\boldsymbol{X}$ 的**奇异值分解**（SVD）。结果就是，最优的 $\boldsymbol{w}$ 就是 $\boldsymbol{X}$ 的主要左奇异向量，也就是 $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ 的主特征向量。投影数据就称为原始数据的主成分。

6.2.5 主成分的通用解

这个过程是可重复的。一旦找到了第一个主成分，就可以重新运行公式 6-7，只是要加上一个附加条件，即新向量与前面找到的向量是正交的（见公式 6-8）。

公式 6-8 第 $k+1$ 个主成分的目标函数

$$\max_{\boldsymbol{w}} \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w}\text{，其中 } \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w} = 1 \text{ 且 } \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w}_1 = \cdots = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{w}_k = 0$$

解是 $\boldsymbol{X}$ 的第 $k+1$ 个左奇异向量，按照奇异值降序排列。因此，前 k 个主成分对应于 $\boldsymbol{X}$ 的前 k 个左奇异向量。

6.2.6　特征转换

一旦找到了主成分，就可以使用线性投影对特征进行转换。令 $\boldsymbol{X}=\boldsymbol{U\Sigma V}^{\mathrm{T}}$ 是 $\boldsymbol{X}$ 的奇异值分解，且 $\boldsymbol{V}_k$ 是列中包含前 k 个左奇异向量的矩阵。$\boldsymbol{X}$ 的维度是 $n\times d$，其中 d 是初始特征的数量，$\boldsymbol{V}_k$ 的维度是 $d\times k$。除了像公式 6-2 中那样在单向量上进行投影，还可以使用投影矩阵在多个向量上同时进行投影（见公式 6-9）。

公式 6-9　PCA 投影矩阵

$$\boldsymbol{W}=\boldsymbol{V}_k$$

投影坐标矩阵很容易计算，而且可以通过奇异向量彼此正交这一性质进一步简化（见公式 6-10）。

公式 6-10　简单 PCA 转换

$$\boldsymbol{Z}=\boldsymbol{XW}=\boldsymbol{XV}_k=\boldsymbol{U\Sigma V}^{\mathrm{T}}\boldsymbol{V}_k=\boldsymbol{U}_k\boldsymbol{\Sigma}_k$$

投影值就是前 k 个右奇异向量乘以前 k 个奇异值。因此，PCA 的一整套解、成分和投影都可以方便地通过 $\boldsymbol{X}$ 的奇异值分解得出。

6.2.7　PCA实现

在很多 PCA 推导中，首先要对数据进行中心化，然后进行协方差矩阵的特征值分解。但 PCA 最容易的实现方法是对中心化后的数据矩阵进行奇异值分解。

PCA 实现步骤

(1) 数据矩阵中心化：

$$\boldsymbol{C}=\boldsymbol{X}-\mathbf{1}\boldsymbol{\mu}^{\mathrm{T}}$$

其中 $\mathbf{1}$ 是个全 1 列向量，$\boldsymbol{\mu}$ 是由 $\boldsymbol{X}$ 中每行的平均值组成的列向量。

(2) 计算 SVD：

$$\boldsymbol{C}=\boldsymbol{U\Sigma V}^{\mathrm{T}}$$

(3) 找出主成分。前 k 个主成分是 $\boldsymbol{V}$ 的前 k 列，也就是对应于 k 个最大奇异值的右奇异向量。

(4) 转换数据。转换后的数据就是 $\boldsymbol{U}$ 的前 k 列。（如果需要白化，就用奇异值的倒数乘以向量。这要求选择的奇异值不是 0。参见 6.4 节。）

6.3 PCA实战

让我们通过在图像数据上应用 PCA，更好地理解一下 PCA。MNIST 数据集中包含了从 0 到 9 这几个数字的手写体的图像，初始图像是 28 像素 × 28 像素。scikit-learn 中有个分辨率更低的图像子集，每张图像被下采样为 8 像素 × 8 像素。scikit-learn 中的初始数据有 64 个维度。在例 6-1 中，我们要使用 PCA，并使用前 3 个主成分对数据集进行可视化。

例 6-1 scikit-learn 数字数据集（MNIST 数据集的一个子集）的主成分分析

```
>>> from sklearn import datasets
>>> from sklearn.decomposition import PCA

# 载入数据
>>> digits_data = datasets.load_digits()
>>> n = len(digits_data.images)

# 每张图像被表示为一个8×8数组。将数组扁平化，作为PCA的输入
>>> image_data = digits_data.images.reshape((n, -1))
>>> image_data.shape
(1797, 64)

# 数字的真实值标签在每张图片中
>>> labels = digits_data.target
>>> labels
array([0, 1, 2, ..., 8, 9, 8])

# 为这个数据集拟合一个PCA转换器
# 自动选择主成分的数目，使主成分能解释至少80%原来的方差
>>> pca_transformer = PCA(n_components=0.8)
>>> pca_images = pca_transformer.fit_transform(image_data)
>>> pca_transformer.explained_variance_ratio_
array([ 0.14890594,  0.13618771,  0.11794594,  0.08409979,  0.05782415,
        0.0491691 ,  0.04315987,  0.03661373,  0.03353248,  0.03078806,
        0.02372341,  0.02272697,  0.01821863])
>>> pca_transformer.explained_variance_ratio_[:3].sum()
0.40303958587675121

# 结果可视化
>>> import matplotlib.pyplot as plt
>>> from mpl_toolkits.mplot3d import Axes3D
>>> %matplotlib notebook
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, projection='3d')
>>> for i in range(100):
...     ax.scatter(pca_images[i,0], pca_images[i,1], pca_images[i,2],
...                marker=r'${}$'.format(labels[i]), s=64)

>>> ax.set_xlabel('Principal component 1')
>>> ax.set_ylabel('Principal component 2')
>>> ax.set_zlabel('Principal component 3')
```

图 6-3 中给出了前 100 个投影图像的 3D 绘图，其中的标记对应于图像标签。前 3 个主成

分解释了数据集中大约 40% 的总方差，这个结果绝对算不上很好，但它可以让我们在低维度上很方便地进行可视化。可以看到，PCA 可以将相似的数字紧密地组织在一起。0 和 6 位于同一区，1 和 7、3 和 9 也是同样的情况。空间被大致划分为 0、4、6 在一侧，其余数字在另一侧。

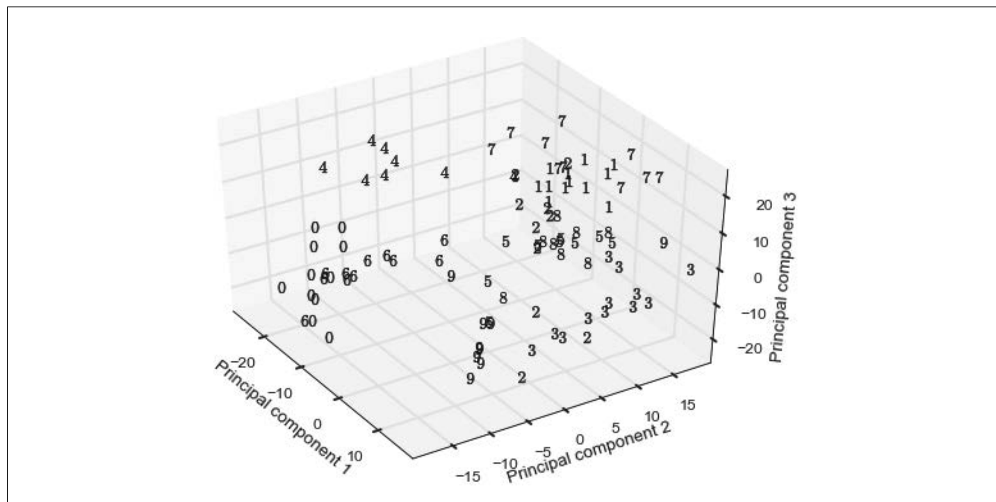


图 6-3：MNIST 数据子集的 PCA 投影——标记对应于图像标签

因为数字之间还有相当数量的重叠，所以在投影空间中使用线性分类器将数字区分开来还是很困难的。因此，如果我们的任务是区分手写数字，而且选择的模型是线性分类器的话，那么只使用前 3 个主成分作为特征是不够的。尽管如此，看看一个 64 维的数据集是如何被转换到三维空间中还是很有趣的。

6.4　白化与ZCA

由于目标函数的正交限制，PCA 转换有一个非常好的副作用：转换后的特征都是不相关的。换句话说，每对特征向量之间的内积都是 0。使用奇异向量的正交性质可以很容易地进行证明：

$$\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z} = \boldsymbol{\Sigma}_k\boldsymbol{U}_k^{\mathrm{T}}\boldsymbol{U}_k\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}_k^2$$

结果是个对角阵，对角线上是奇异值的平方，表示每个特征向量与自己的相关度，也称为 ℓ^2 范数。

有时候，还应该通过归一化把特征的长度变为 1。在信号处理领域，这种操作称为**白化**。这样做可以得到一组特征，彼此之间的相关度为 0，与自身的相关度为 1。数学上，将 PCA 转换乘以奇异值的倒数，就可以实现白化（见公式 6-11）。

公式 6-11　PCA+ 白化

$$\boldsymbol{W}_{\text{white}} = \boldsymbol{V}_k \boldsymbol{\Sigma}_k^{-1}$$

$$\boldsymbol{Z}_{\text{white}} = \boldsymbol{X}\boldsymbol{V}_k \boldsymbol{\Sigma}_k^{-1} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}_k \boldsymbol{\Sigma}_k^{-1} = \boldsymbol{U}_k$$

白化与数据降维是彼此独立的，可以分别进行。例如，**零相位成分分析**（zero-phase component analysi，ZCA）（Bell and Sejnowski, 1996）是一种与 PCA 联系非常紧密的白化转换，但它不会减少特征的数量。ZCA 白化使用未削减的整个主成分集合，还要再乘上一个 $\boldsymbol{V}^{\mathrm{T}}$（见公式 6-12）。

公式 6-12　ZCA 白化

$$\boldsymbol{W}_{\text{ZCA}} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{V}^{\mathrm{T}}$$

$$\boldsymbol{Z}_{\text{ZCA}} = \boldsymbol{X}\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{V}^{\mathrm{T}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}\boldsymbol{V}\boldsymbol{\Sigma}^{-1} = \boldsymbol{U}$$

简单的 PCA 投影（见公式 6-10）可以在新特征空间中以主成分为基生成坐标。这些坐标只表示投影向量的长度，不表示方向。乘以主成分之后，才能得到长度和方向。另一个合理解释是，这次相乘可以将坐标旋转回初始特征空间。（$\boldsymbol{V}$ 是一个正交矩阵，正交矩阵可以对输入进行没有拉伸和压缩的旋转。）所以，ZCA 生成的白化数据与初始数据是最接近的（用欧氏距离衡量）。

6.5　PCA的局限性与注意事项

在使用 PCA 进行数据降维时，必须确定要使用的主成分的数目（k）。和所有超参数一样，这个数目可以根据最终模型的质量来进行优化，但也有一些不需要昂贵计算成本的启发式方法。

选择 k 的一种可行方法是要求主成分能解释一定比例的总方差。（scikit-learn 的 `PCA` 包中有这个选项。）在 k 个主成分上的投影的方差为：

$$\|\boldsymbol{X}\boldsymbol{v}_k\|^2 = \|\boldsymbol{u}_k \sigma_k\|^2 = \sigma_k^2$$

这个方差就是 $\boldsymbol{X}$ 第 k 大奇异值的平方。奇异值的排序列表称为矩阵的**谱**（spectrum）。因此，要确定使用多少主成分，可以对数据矩阵做一个简单的谱分析，并选定能解释足够方差的阈值。



根据解释的方差选择 k

要保留足够的主成分来解释数据中 80% 的总方差，可以这样选择 k：

$$\frac{\sum_{i=1}^{k}\sigma_i^2}{\sum_{i=1}^{d}\sigma_i^2} \geqslant 0.8$$

另一种选择 k 的方法涉及数据集的本征维数。这是个非常模糊的概念，但也可以通过矩阵的谱来确定。简单地说，如果谱中包含一些非常大的奇异值和一些非常小的奇异值，我们就可以只保留那些非常大的奇异值，丢弃其余奇异值。有时候，谱中其余的奇异值不是非常小，但头部和尾部的值之间有比较大的缺口，这也是一个非常合理的界限。这种方法需要对谱进行人工观察，因此不能作为自动流程的一部分。

对 PCA 的一种主要诟病是转换过程太复杂了，而且由此得到的结果也难以解释。主成分和投影向量是实数值，可能是正的，也可能是负的。主成分实质上是（中心化后的）行的线性组合，投影值则是列的线性组合。例如，在一个股票收益应用中，每个因子都是股票收益时间片的一个线性组合。其中的含义呢？很难用人类可以理解的理由来解释这些学习得出的因子。因此，分析师很难相信这些结果。如果不能解释为什么应该把成千上万其他人的钱投到一支特定的股票上，你可能就不会使用这个模型。

PCA 的计算成本是非常昂贵的，它依赖于 SVD，而 SVD 就是个对计算能力要求非常高的过程。要计算出一个矩阵的完整 SVD，需要 $O(nd^2 + d^3)$ 次操作（Golub and Van Loan, 2012），假设 $n \geqslant d$，即数据点数量大于特征数量。尽管我们只需要 k 个主成分，计算截断后的 SVD（k 个最大奇异值及其对应的奇异向量）仍然需要 $O((n+d)^2k) = O(n^2k)$ 次操作。当有大量数据点和特征时，计算成本令人望而却步。

在流式数据、批量更新或完整数据的抽样中，是难以执行 PCA 操作的。SVD 的流式计算、SVD 更新，以及根据子样本计算 SVD，都是非常困难的研究问题。算法是存在的，但代价是降低了准确率。其中暗含的一个意思就是，当把测试数据投影到训练过程中找出的主成分时，我们只能期待较低的表示准确率。随着数据分布的改变，我们必须重新计算当前数据集的主成分。

最后，最好不要对原始计数（单词计数、音乐播放计数、电影观看计数等）使用 PCA。原因在于这样的计数通常包含巨大的异常值。（很有可能一个狂热的影迷观看了《指环王》314 582 次，这会使其他计数相形见绌。）正如我们所知，PCA 找寻的是特征之间的线性相关度。相关度和方差统计量对大的异常值非常敏感，一个巨大的异常值就可以显著改变这些统计量。所以，先清理数据中的异常值（3.2.2 节）是非常好的一种做法，也可以使用像 tf-idf（第 4 章）或对数变换（2.3 节）这样的缩放转换。

6.6　用例

通过找出特征之间的线性相关模式，PCA 可以减少特征空间的维度。因为要用到 SVD，所以对于数量超过几千的特征，PCA 的计算成本是非常昂贵的。但对于数量较少的实数值特征，它还是非常值得尝试的。

PCA 转换丢弃了数据中的一些信息，因此，下游模型的训练成本更低，但精确度也下降了。在 MNIST 数据集上，有人发现使用通过 PCA 得到的降维数据会导致分类模型的精确

度降低。在这种情况下，使用 PCA 的效果是喜忧参半的。

PCA 最精彩的应用之一是在时间序列中进行异常检测。Lakhina 等人使用 PCA 在互联网流量中检测和诊断异常。他们重点关注流量异常，即什么时候从一个网络区域到另一个网络区域的流量会发生剧烈的变化。这些突变可能就是网络错误配置或拒绝服务攻击的信号，不管是哪一种情况，知道这种变化在什么时候和在哪里发生对于互联网管理员来说都是非常有价值的。

因为互联网上的流量太多了，所以小型区域内的孤立突变很难探测，但多数流量都是在一个相对较小的主干链接集合中处理的。关键在于，流量异常会同时影响多个链接（因为数据包要经过多个节点才能到达它们的目标）。我们可以将每个链接作为一个特征，并将每个时间段的流量作为测量值，那么一个数据点就是在一个时间片内通过网络上所有链接的流量的测量。这种矩阵的主成分可以表明网络上的整体流量趋势，余下的成分表示还有残差信号，其中就包括异常。

PCA 还经常应用在金融建模中。在这些用例中，PCA 被作为一种**因子分析**。因子分析是一个统计方法族，它们致力于使用少量潜在的因子来描述数据中可见的变异。在因子分析应用中，分析目标是找到可解释的成分，不是对数据进行转换。

像股票回报这样的金融数量指标通常是彼此相关的。一些股票经常同时上涨或下跌（正相关），也可能沿着相反的方向变化（负相关）。为了平衡这种波动并降低风险，一个投资组合需要一组分散的、彼此不相关的股票。（如果篮子可能沉到水底，就不要把所有鸡蛋放在一个篮子里。）找出强相关模式有助于确定投资策略。

股票相关模式可以是行业范围内的。例如，科技股的价格会同时上升和下跌，而石油股的价格上升时，航空股会下跌。但行业可能不是解释这个结果的最好方式，分析师们还会在可知的统计量中寻求意料之外的相关性。特别地，为了找出通常同时变化的股票，**统计因子模型**（Connor, 1995）在单支股票回报的时间序列矩阵上运行 PCA。在这个用例中，最终目标是找到主成分本身，而不是转换数据。

当从图像中学习时，在预处理阶段可以使用 ZCA。在自然图像中，邻接像素通常具有相似的颜色，ZCA 白化可以除去这种相关性，从而使随后的建模工作可以集中在更有趣的图像结构上。Krizhevsky (2009) 的论文“Learning Multiple Layers of Features from Images”中给出了很多精彩的例子，说明了 ZCA 白化在自然图像处理中的作用。

很多深度学习模型使用 PCA 或 ZCA 作为预处理的一个步骤，但它并不总是必要的。在“Factored 3-Way Restricted Boltzmann Machines for Modeling Natural Images”一文中，Ranzato 等人认为“白化并不是必要的步骤，但可以加快算法收敛”。在“An Analysis of Single-Layer Networks in Unsupervised Feature Learning”中，Coates 等人发现 ZCA 白化对某些模型有帮助，但不是对所有模型都有帮助。（请注意，这篇文章中的模型是无监督特征学习模型，所以在其他特征工程方法中可以使用一下 ZCA。在机器学习流程中，方法的堆叠和链接是很常见的。）

6.7 小结

关于 PCA 的讨论到此结束。关于 PCA 我们应该记住的主要两点是它的原理（线性投影）和目标（最大化投影数据的方差）。PCA 的解要用到协方差矩阵的特征值分解，它与数据矩阵的 SVD 联系非常紧密。还可以这样形象地理解 PCA：将数据挤入一张尽量松软的薄饼中。

PCA 是一种模型驱动的特征工程方法。（只要看到一个目标函数，就应该马上猜想它的背后可能隐藏着一个模型。）其中的建模假设是方差能够很好地表示数据中包含的信息。同样，模型要找出的是特征之间的线性相关性。在若干应用中可以使用 PCA 来减少相关性或找出输入中常见的因子。

PCA 是一种广为人知的数据降维方法，但它也有自身的局限性，比如高昂的计算成本和不能解释的结果。它适合应用在预处理阶段，特别是特征之间存在线性相关性的时候。

当作为一种消除线性相关性的方法时，PCA 还可以和白化方法结合使用。它的“表亲”ZCA 可以用可解释的方式对数据进行白化，但不能降低数据维度。

6.8 参考文献

Bell, Anthony J. and Terrence J. Sejnowski. Edges Are the“Independent Components”of Natural Scenes [G]. Advances in Neural Information Processing Systems 9 (1996): 831–837.

Coates, Adam, Andrew Y. Ng, and Honglak Lee. An Analysis of Single-Layer Networks in Unsupervised Feature Learning [C]. Proceedings of the 14th International conference on Artificial Intelligence and Statistics (2011): 215–223.

Connor, Gregory. The Three Types of Factor Models: A Comparison of Their Explanatory Power [J]. Financial Analysts Journal 51:3 (1995) 42–46.

Golub, Gene H., and Charles F. Van Loan. Matrix Computations [M]. 4th ed. Baltimore, MD: Johns Hopkins University Press, 2012.

Krizhevsky, Alex. Learning Multiple Layers of Features from Tiny Images [D]. MSc thesis, University of Toronto, 2009.

Lakhina, Anukool, Mark Crovella, and Christophe Diot. Diagnosing Network-wide Traffic Anomalies [C]. Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (2004): 219–230.

Ranzato, Marc’Aurelio, Alex Krizhevsky, and Geoffrey E. Hinton. Factored 3-Way Restricted Boltzmann Machines for Modeling Natural Images [C]. Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (2010): 621–628.

第 7 章

非线性特征化与*k*-均值模型堆叠

当数据位于一个薄饼状的线性子空间时，PCA 是非常有用的。但如果数据形成了一个更加复杂的形状，情况又将如何呢？[1] 平面（线性子空间）可以推广为**流形**（非线性子空间），可以把流形看作一个可以以多种方式伸展和卷动的曲面。[2]

如果线性子空间是一张平展的纸，那么非线性流形的一个简单例子就是卷起来的纸，它有个非正式的名称，叫作**瑞士卷**（见图 7-1）。一旦卷了起来，二维平面就占据了三维空间，尽管它本质上还是个二维对象。换句话说，它具有低本征维数，这个概念我们在 6.1 节中已经接触过了。如果能够以某种方式展开瑞士卷，就可以恢复二维平面。这就是**非线性数据降维**的目标，它假定流形要比它所在的全维度空间简单，然后试图将其展开。

流形能够展开的关键在于，即使一个大的流形看上去非常复杂，但它的每个数据点的邻近区域通常可以非常好地近似为一块平面。换句话说，可以通过多个小平面使用局部结构组成全局结构。[3] 非线性数据降维也称为**非线性嵌入**或**流形学习**。非线性嵌入可以非常有效地

注 1：本章的写作灵感来自于和 Ted Dunning 的一段对话，他是一位活跃的 Apache 贡献者和知名作家。本章中的堆叠示例就是 Ted 提供的，他还对本章的写作方法提供了很多有益的建议。如果有人想找个合著者写作一些单独的章节，那么 Ted 就是你应该找的人。

注 2：在本章中，我们将交替使用“曲面”和“流形”这两个词。我们可以很好地类推到嵌入在三维空间中的二维流形，但超过三维之后就不可行了。高维流形不符合我们对“曲面”的通常理解。有些非常奇异的流形有洞，还有些流形会以现实世界中根本不会存在的方式环回到自己本身（如 M.C. Escher 的无尽瀑布）。多数数据模型的假设是性质良好的流形，而不是那些稀奇古怪的流形。

注 3：这是数学中的一种“实证有效”思想。例如，用函数的导数测量每个点处的速度变化。全局上，函数可以是任意的形式，但在局部，它可以近似为导数的线性函数。如果知道了每个点上的导数，那么就可以使用微积分或多或少地还原整个初始函数。

将高维数据压缩为低维数据，常用于二维空间或三维空间中的可视化。

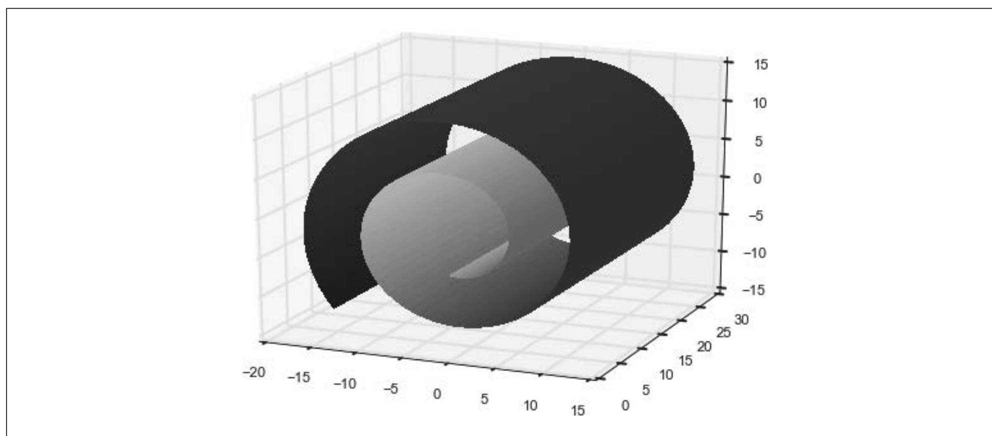


图 7-1：瑞士卷，一个非线性流形

但是，尽量降低特征维度只是特征工程目标的一小部分，它的根本目标还是为当前任务找到**正确**的特征。在本章中，正确的特征是那些能表示出数据的空间特性的特征。

聚类算法通常不被用作局部结构学习技术，但实际上它完全可以胜任。彼此相近（可以用一种特定的度量方式来定义“近”的概念）的点属于同一个簇。给定一个聚类，数据点可以用它的簇成员向量来表示。如果簇的数量小于初始的特征数量，那么相对于初始表示，这种新表示就具有更少的维度，初始数据就被压缩进一个更低维度的空间。本章将解释这种思想。

与非线性嵌入技术相比，聚类会生成更多特征。但如果最终目标是特征工程，而不是可视化，这就不是问题了。

我们将通过一种称为 *k*-均值的常用聚类算法来说明局部结构学习的思想，这种方法简单易行。与其说 *k*-均值方法的作用是非线性流形降维，还不如说它执行了**非线性流形特征提取**。使用正确的话，*k*-均值聚类可以成为特征工程的一项神兵利器。

7.1 *k*-均值聚类

k-均值是一种聚类算法。聚类算法根据数据在空间中的分布方式为其分组。聚类是一种**非监督**学习方法，它不需要任何形式的标签——这种算法的目的就是仅基于数据本身的结构推测出簇标签。

聚类算法依赖于**度量方式**，即对数据点之间相近程度的测量。最常用的度量方式是欧氏距离，或称欧几里得度量，它来自于欧氏几何，测量的是两点之间的直线距离。这种度量方

式对我们来说非常正常，因为这就是现实世界中随处可见的距离。

两个向量 x 和 y 之间的欧氏距离是 $x - y$ 的 ℓ^2 范数。（想更多地了解 ℓ^2 范数，参见 2.4.3 节。）使用数学语言描述的话，它通常写作 $\|x-y\|_2$，或者就是 $\|x-y\|$。

k-均值会建立一种硬聚类，也就是说每个数据点都属于且只属于一个簇。该算法通过学习来定位簇中心点，使得每个数据点和簇中心点之间的欧氏距离的总和最小。对于那些喜欢数学公式胜过文字描述的人，可以看看下面的目标函数：

$$\min_{C_1,\cdots,C_k,\mu_1,\cdots,\mu_k} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|_2$$

每个簇 C_i 包含数据点的一个子集，簇中心点 i 等于簇中所有数据点的平均值：

$$\mu_i = \sum_{x \in C_i} x / n_i$$

其中 n_i 表示簇 i 中数据点的数量。

图 7-2 展示了在两种不同的、随机生成的数据集上的 k-均值聚类。(a) 中的数据是使用几种随机高斯分布生成的，这些分布的方差相同但均值不同。(c) 中的数据则是完全随机生成的。这些用于实验的问题非常容易解决，k-均值聚类的效果非常好。（聚类结果对簇的数目非常敏感，簇数目必须在算法中给定。）

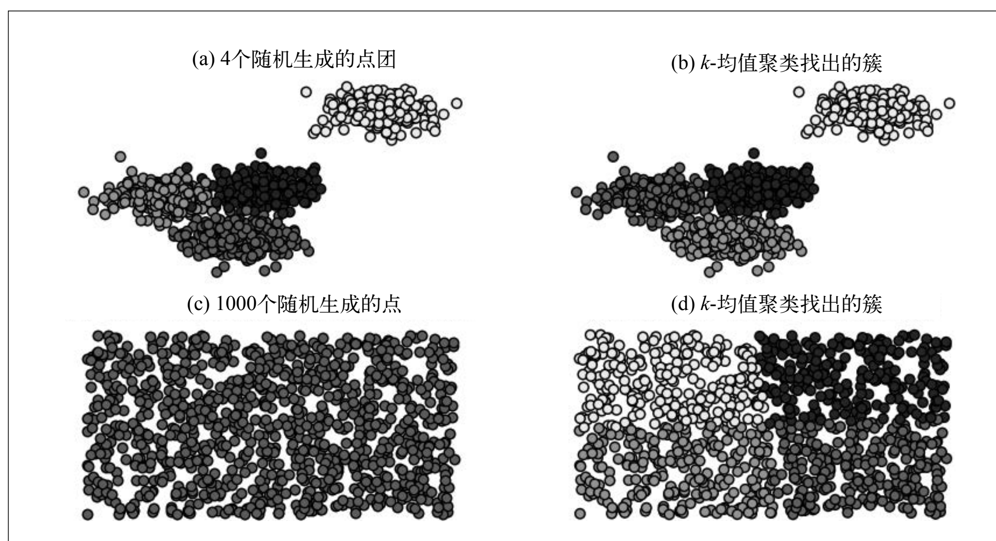


图 7-2：演示聚类算法如何划分空间的 k-均值示例

本例使用的代码见例 7-1。

例 7-1　生成 k-均值示例的代码

```
>>> import numpy as np
>>> from sklearn.cluster import KMeans
>>> from sklearn.datasets import make_blobs

>>> import matplotlib.pyplot as plt

>>> n_data = 1000
>>> seed = 1
>>> n_clusters = 4

# 生成符合高斯随机分布的点团，并运行k-均值算法
>>> blobs, blob_labels = make_blobs(n_samples=n_data, n_features=2,
...                                  centers=n_centers, random_state=seed)
>>> clusters_blob = KMeans(n_clusters=n_centers, random_state=seed).fit_
predict(blobs)

# 生成完全随机的数据，并运行k-均值算法
>>> uniform = np.random.rand(n_data, 2)
>>> clusters_uniform = KMeans(n_clusters=n_clusters,
...                           random_state=seed).fit_predict(uniform)

# 对结果进行可视化的Matplotlib代码
>>> figure = plt.figure()
>>> plt.subplot(221)
>>> plt.scatter(blobs[:, 0], blobs[:, 1], c=blob_labels, cmap='gist_rainbow')
>>> plt.title("(a) Four randomly generated blobs", fontsize=14)
>>> plt.axis('off')

>>> plt.subplot(222)
>>> plt.scatter(blobs[:, 0], blobs[:, 1], c=clusters_blob, cmap='gist_rainbow')
>>> plt.title("(b) Clusters found via K-means", fontsize=14)
>>> plt.axis('off')

>>> plt.subplot(223)
>>> plt.scatter(uniform[:, 0], uniform[:, 1])
>>> plt.title("(c) 1000 randomly generated points", fontsize=14)
>>> plt.axis('off')

>>> plt.subplot(224)
>>> plt.scatter(uniform[:, 0], uniform[:, 1], c=clusters_uniform, cmap='gist_
rainbow')
>>> plt.title("(d) Clusters found via K-means", fontsize=14)
>>> plt.axis('off')
```

7.2　使用聚类进行曲面拼接

一般的聚类应用假定在数据中可以找到自然形成的簇，也就是说，空间中分布着一些密集数据区域，空间的其他部分则相对空旷。在这种情况下，需要确定正确的簇数目，而为了选择这个 k，人们发明了聚类索引来测量数据分组的质量。

但是，当数据分布得较为均匀，就像图 7-2c 中那样的时候，就没有一个正确的簇数目了。这时，聚类算法的作用就是**矢量量化**，即将数据划分为有限数目的数据段。当使用量化矢量而不是原始矢量时，可以基于可接受的近似误差来选择簇的数目。

形象地说，这种 *k*-均值聚类的使用方式就像是用补丁来覆盖数据曲面，如图 7-3 所示。如果在瑞士卷数据集上运行 *k*-均值聚类，这也就是我们能实际得到的结果。

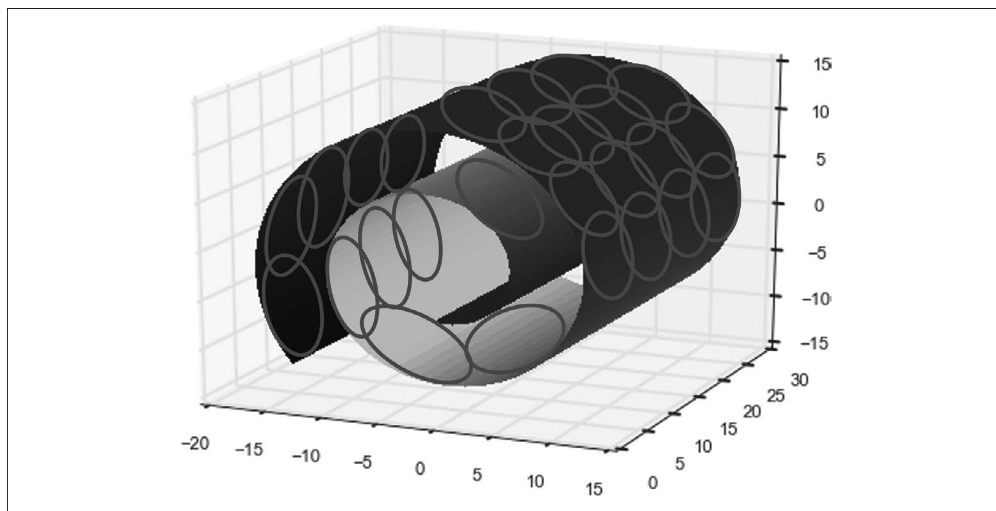


图 7-3：通过聚类算法在瑞士卷上得到的概念性局部补丁

例 7-2 使用 scikit-learn 在瑞士卷上生成了一个带噪声的数据集，对其使用 *k*-均值算法进行了聚类，并使用 Matplotlib 对聚类结果进行了可视化。数据点按照它们的簇 ID 进行了着色。

例 7-2　瑞士卷上的 *k*-均值聚类

```
>>> from mpl_toolkits.mplot3d import Axes3D
>>> from sklearn import manifold, datasets

# 生成带噪声的瑞士卷数据集
>>> X, color = datasets.samples_generator.make_swiss_roll(n_samples=1500)

# 使用100个k-均值簇对数据进行近似
>>> clusters_swiss_roll = KMeans(n_clusters=100, random_state=1).fit_predict(X)

# 使用k-均值簇ID作为颜色来绘制数据集
>>> fig2 = plt.figure()
>>> ax = fig2.add_subplot(111, projection='3d')
>>> ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=clusters_swiss_roll, cmap='Spectral')
```

在这个例子中，我们在瑞士卷曲面上生成了 1500 个随机的数据点，并指定 *k*-均值通过 100 个簇来对数据进行近似。使用 100 作为簇的数目，是因为它看上去足够大，完全可以覆盖这么一个相对较小的空间。结果（见图 7-4）看上去非常不错，簇确实分布在局部，流形

的不同部分也对应着不同的簇。棒极了！你做到了吗？

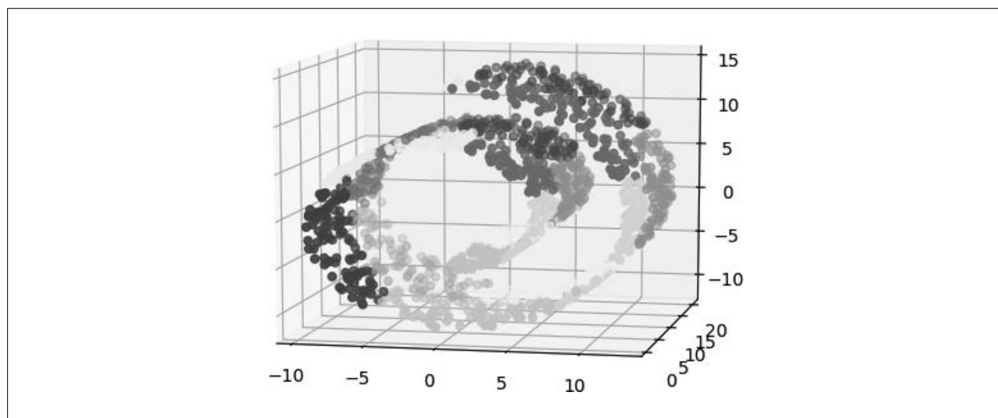


图 7-4：使用 *k*-均值通过 100 个簇来近似瑞士卷数据集

问题是，如果选择的 *k* 值过小，从流形学习的角度来说，结果就不会太好。图 7-5 展示了在瑞士卷上使用 10 个簇的 *k*-均值结果。可以清楚地看到，很多来自于流形中明显不同部分的数据被映射到了同一个簇中（比如黄色、紫色、绿色和品红色的簇。看，我们说过最好使用彩色图形来演示的！）。

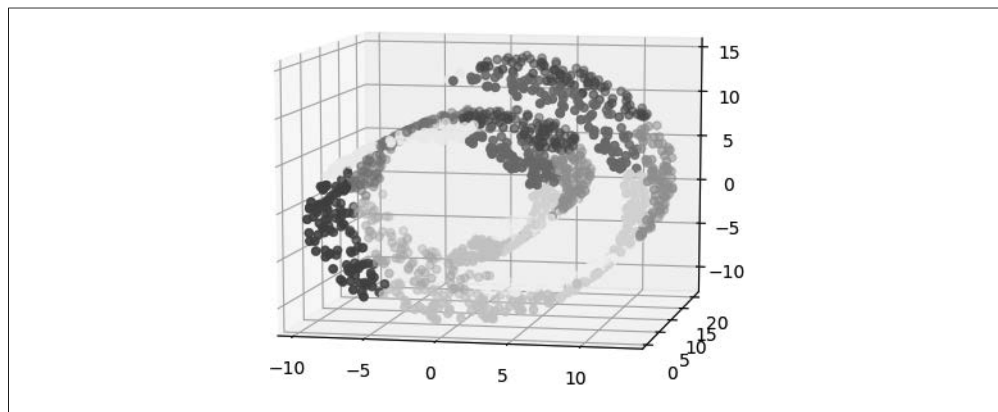


图 7-5：瑞士卷上的 10 簇 *k*-均值聚类

如果数据均匀一致地分布在整个空间中，那么如何选择正确的 *k* 就可以归结为一个球体填充问题。在 *d* 维空间中，我们大致可以拟合出 $1/r^d$ 个半径为 *r* 的球体。每个 *k*-均值簇都是一个球体，球体半径是球体中的点和中心点之间的最大误差。所以，如果可以容忍每个数据点的最大近似误差是 *r* 的话，那么簇的数目就是 $O(1/r^d)$，其中 *d* 是数据初始特征空间的维度。

均匀分布是 *k*-均值算法的最坏情况。如果数据密度是不均匀的，那么就可以使用较少的簇表示更多数据。通常，很难说数据在高维空间中是如何分布的。有人很保守，会选择一个更大的 *k*，但 *k* 也不能过大，因为 *k* 会成为后续建模阶段的特征数量。

7.3 用于分类问题的*k*-均值特征化

当使用 *k*-均值作为特征化技术时，一个数据点可以通过它在簇中的隶属关系进行表示（簇隶属关系分类变量的稀疏 one-hot 编码，参见 5.1.1 节），这将在下面进行说明。

如果还有目标变量，那么也可以将目标变量中的信息作为聚类过程的提示。加入目标变量信息的一种方法是，直接将目标变量作为 *k*-均值算法的一个额外输入特征。因为我们的目标是将所有输入维度上的欧氏距离的总和最小化，所以聚类过程除了在初始特征空间中以外，也会在目标变量值之间平衡相似度。可以对目标变量进行缩放，来增加或减少聚类算法对它的关注程度。目标变量之间的差距较大时，会生成更加注重分类边界的簇。



_k_-均值特征化

聚类算法可以分析数据的空间分布，因此，*k*-均值特征化可以生成数据的压缩空间索引，供下一阶段的模型使用。这就是**模型堆叠**的一个例子。

例 7-3 展示了一个简单的 *k*-均值特征生成器，它定义了一个类对象，可以拟合训练数据并转换新数据。

例 7-3　*k*-均值特征生成器

```
>>> import numpy as np
>>> from sklearn.cluster import KMeans

>>> class KMeansFeaturizer:
...     """Transforms numeric data into k-means cluster memberships.
...
...     This transformer runs k-means on the input data and converts each data point
...     into the ID of the closest cluster. If a target variable is present, it is
...     scaled and included as input to k-means in order to derive clusters that
...     obey the classification boundary as well as group similar points together.
...     """
...
...     def __init__(self, k=100, target_scale=5.0, random_state=None):
...         self.k = k
...         self.target_scale = target_scale
...         self.random_state = random_state
...
...     def fit(self, X, y=None):
...         """Runs k-means on the input data and finds centroids.
...         """
...         if y is None:
...             # 没有目标变量，就执行普通k-均值算法
```

```
...             km_model = KMeans(n_clusters=self.k,
...                               n_init=20,
...                               random_state=self.random_state)
...             km_model.fit(X)
...
...             self.km_model_ = km_model
...             self.cluster_centers_ = km_model.cluster_centers_
...             return self
...
...         # 有目标信息，使用恰当的缩放
...         # 并将其包含在k-均值算法的输入数据中
...         data_with_target = np.hstack((X, y[:,np.newaxis]*self.target_scale))
...
...         # 在数据和目标变量上建立一个预训练k-均值模型
...         km_model_pretrain = KMeans(n_clusters=self.k,
...                                    n_init=20,
...                                    random_state=self.random_state)
...         km_model_pretrain.fit(data_with_target)
...
...         # 第二次运行k-均值算法，得到不带目标变量信息的初始空间中的簇。
...         # 使用在预训练中得到的中心点进行初始化。
...         # 执行一次迭代，进行簇分配和中心点重计算。
...         km_model = KMeans(n_clusters=self.k,
...                           init=km_model_pretrain.cluster_centers_[:,:2],
...                           n_init=1,
...                           max_iter=1)
...         km_model.fit(X)
...
...         self.km_model = km_model
...         self.cluster_centers_ = km_model.cluster_centers_
...         return self
...
...     def transform(self, X, y=None):
...         """Outputs the closest cluster ID for each input data point.
...         """
...         clusters = self.km_model.predict(X)
...         return clusters[:,np.newaxis]
...
...     def fit_transform(self, X, y=None):
...         self.fit(X, y)
...         return self.transform(X, y)
```

在例 7-4 的聚类中，为了演示一下使用目标信息和不使用目标信息之间的区别，我们使用 scikit-learn 的 `make_moons` 函数生成一个人造数据集，再对其应用前面的特征生成器，并绘制出簇边界的 Voronoi 图。

例 7-4　使用目标提示和不使用目标提示的 *k*-均值特征化

```
>>> from scipy.spatial import Voronoi, voronoi_plot_2d
>>> from sklearn.datasets import make_moons

>>> training_data, training_labels = make_moons(n_samples=2000, noise=0.2)
>>> kmf_hint = KMeansFeaturizer(k=100, target_scale=10).fit(training_data,
```

```
...                                                       training_labels)
>>> kmf_no_hint = KMeansFeaturizer(k=100, target_scale=0).fit(training_data,
...                                                         training_labels)

>>> def kmeans_voronoi_plot(X, y, cluster_centers, ax):
...     """Plots the Voronoi diagram of the k-means clusters overlaid with the
data"""
...     ax.scatter(X[:, 0], X[:, 1], c=y, cmap='Set1', alpha=0.2)
...     vor = Voronoi(cluster_centers)
...     voronoi_plot_2d(vor, ax=ax, show_vertices=False, alpha=0.5)
```

图 7-6 对比了两种结果。数据集中的两个月亮都按照它们的类标签进行着色。其中下图展示了不使用目标信息训练得出的簇，请注意有一些簇跨域了两个类别之间的空旷空间。上图是聚类算法使用了目标信息后的结果，簇边界沿着类别边界的对齐有了明显的改善。

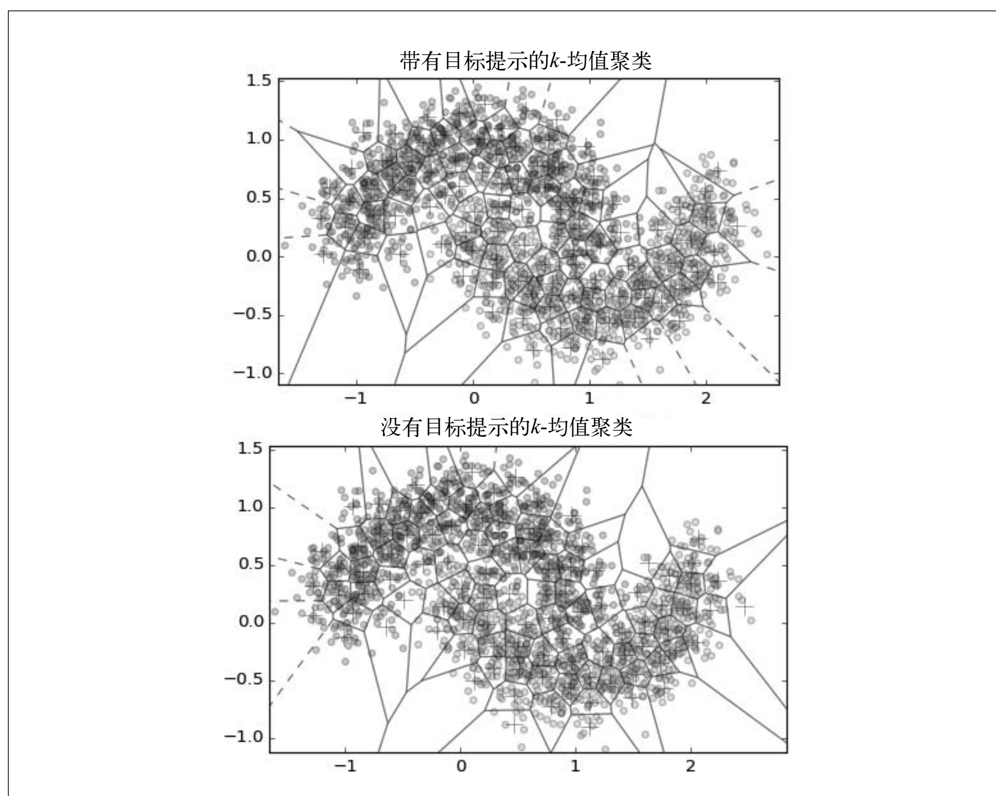


图 7-6：使用目标类信息的 *k*-均值聚类（上）和不使用目标类信息的 *k*-均值聚类（下）

我们测试一下 *k*-均值特征对于分类问题的有效性。在例 7-5 中，我们进行了一次逻辑回归，它的输入数据通过 *k*-均值簇特征进行了增强。我们将它的结果与如下分类方法进行比较，包括使用径向基函数核的支持向量机（RBF SVM）、*k*-最近邻（kNN）、随机森林（RF）和梯度提升树（GBT）。RF 和 GBT 是当前使用非常广泛的非线性分类器，性能一流。RBF

SVM 是非常适合在欧氏空间中使用的非线性分类器。kNN 按照 *k* 个最近邻的平均值对数据进行分类。

分类器的默认输入数据由每个数据点的二维坐标组成，逻辑回归中还使用了表示簇隶属关系的特征（图 7-7 中标记为“LR with k-means”）。我们还进行了仅使用二维坐标的逻辑回归，作为参照基准（标记为“LR”）。

例 7-5　使用 *k*-均值簇特征进行分类

```
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.svm import SVC
>>> from sklearn.neighbors import KNeighborsClassifier
>>> from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

### 使用与训练数据相同的分布生成一些测试数据
>>> test_data, test_labels = make_moons(n_samples=2000, noise=0.3)

### 使用k-均值特征生成器生成簇特征
>>> training_cluster_features = kmf_hint.transform(training_data)
>>> test_cluster_features = kmf_hint.transform(test_data)

### 使用簇特征构造新的输入特征
>>> training_with_cluster = scipy.sparse.hstack((training_data,
...                                              training_cluster_features))
>>> test_with_cluster = scipy.sparse.hstack((test_data, test_cluster_features))

### 建立分类器
>>> lr_cluster = LogisticRegression(random_state=seed).fit(training_with_cluster,
...                                                        training_labels)
>>> classifier_names = ['LR',
...                     'kNN',
...                     'RBF SVM',
...                     'Random Forest',
...                     'Boosted Trees']
>>> classifiers = [LogisticRegression(random_state=seed),
...                KNeighborsClassifier(5),
...                SVC(gamma=2, C=1),
...                RandomForestClassifier(max_depth=5, n_estimators=10, max_
features=1),
...                GradientBoostingClassifier(n_estimators=10, learning_rate=1.0,
...                                           max_depth=5)]
>>> for model in classifiers:
...     model.fit(training_data, training_labels)

### 使用ROC评价分类器性能的辅助函数
>>> def test_roc(model, data, labels):
...     if hasattr(model, "decision_function"):
...         predictions = model.decision_function(data)
...     else:
...         predictions = model.predict_proba(data)[:,1]
...     fpr, tpr, _ = sklearn.metrics.roc_curve(labels, predictions)
...     return fpr, tpr
```

```
### 绘制结果
>>> import matplotlib.pyplot as plt
>>> plt.figure()
>>> fpr_cluster, tpr_cluster = test_roc(lr_cluster, test_with_cluster, test_labels)
>>> plt.plot(fpr_cluster, tpr_cluster, 'r-', label='LR with k-means')

>>> for i, model in enumerate(classifiers):
...     fpr, tpr = test_roc(model, test_data, test_labels)
...     plt.plot(fpr, tpr, label=classifier_names[i])

>>> plt.plot([0, 1], [0, 1], 'k--')
>>> plt.legend()
```

图 7-7 展示了每种分类器在测试集上进行评价时的受试者工作特征（ROC）曲线。ROC 曲线表示的是当改变分类决策边界时真阳性和假阳性之间的权衡。（参见 Zheng (2015) 以获得更多详细信息。）一个好的分类器应该能快速地达到很高的真阳性率和很低的假阳性率，所以，能快速靠近左上角的曲线是极好的。

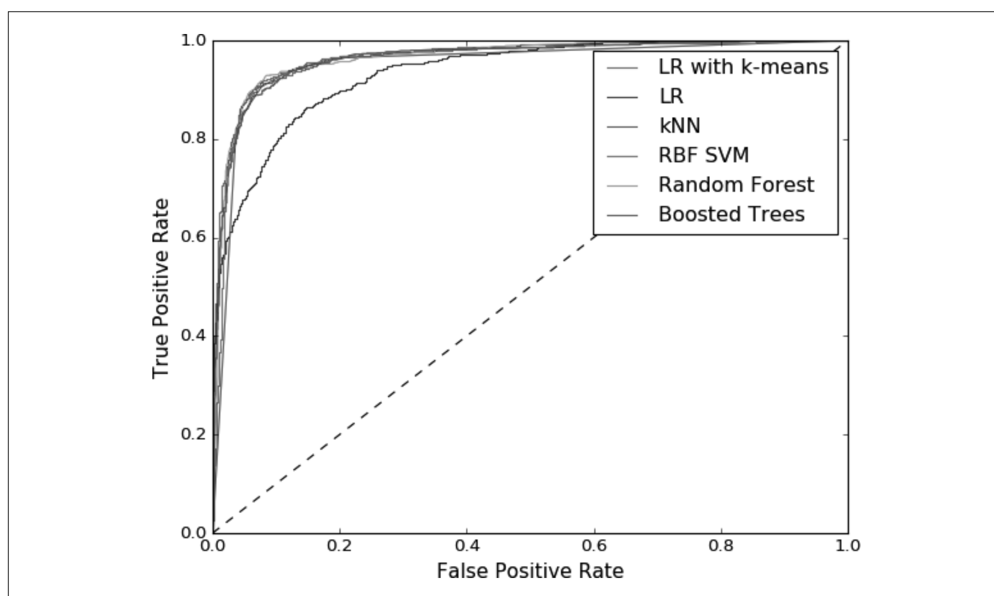


图 7-7：在人造双月数据集上 *k*-means+ 逻辑回归与其他非线性分类器及普通逻辑回归的 ROC 曲线对比

从图中可以看出，使用簇特征的逻辑回归比不使用簇特征的效果要好得多。实际上，使用了簇特征之后，线性分类器的效果与非线性分类器不相上下。在这个非正式的例子中有一个小问题，就是我们没有为任何模型进行超参数调优。如果模型被充分调优，就可能出现性能上的差别。但这个例子至少能够说明，带有 *k*-均值簇特征的逻辑回归模型是可能与非线性分类器相媲美的。这是个不错的结果，因为线性分类器的训练成本要远远低于非线性

分类器。较低的计算成本允许我们在同样的时间段内试验更多带有不同特征的模型，由此增大了得到更好模型的概率。

另一种密集特征化

如果不使用 one-hot 编码的簇隶属关系，还可以使用一个密集向量来表示数据点，这个向量由数据点到每个簇中心点距离的倒数组成。相对于简单的二值化簇分配，这种方式可以保留更多的信息，只不过表示方式是密集的。这就是一种妥协。one-hot 编码的簇隶属关系可以提供一种非常轻量的、稀疏的表示，但需要一个比较大的 k 值来表示形状复杂的数据。距离倒数表示法是密集的，对于建模的各个步骤来说代价更高，但优点是可以使用更小的 k。

稀疏表示和密集表示之间的一种折中方案是，只保留 p 个最近簇的距离倒数。但这样一来，又多了一个需要调优的超参数 p。（现在你能理解为什么特征工程需要考虑这么多细节了吧？）世间没有免费的午餐。

7.4 优点、缺点以及陷阱

使用 k-均值将空间数据转换为特征是**模型堆叠**的一个实例，其中一个模型的输入是另一个模型的输出。另一个堆叠实例是使用决策树类型模型（随机森林或梯度提升树）的输出作为线性分类器的输入。近年来，模型堆叠已经成为了一种越来越流行的技术，因为训练和维护非线性模型的成本非常高昂。堆叠的核心思想是将非线性放入特征中，再使用一种非常简单的、通常是线性的模型作为最后一层。特征生成器可以在线下训练，这意味着我们可以使用昂贵的、需要更多计算能力和内存的模型来生成有用的特征。位于顶层的简单模型可以快速适应在线数据中快速的分布变化。这是一种精确度和速度之间的权衡，这种策略通常使用在像定向广告这样需要快速适应数据分布变化的应用中。



模型堆叠的核心思想

先使用复杂的基础层（通常带有昂贵的模型）生成良好的（通常是非线性的）特征，再与简单快速的顶层模型组合起来。这样通常能实现模型准确率和速度之间的正确平衡。

与使用非线性分类器相比，k-均值与逻辑回归的堆叠模型更容易训练和存储。表 7-1 详细列出了一些机器学习模型在计算能力和内存方面的训练和预测复杂度。n 表示数据点的数量，d 表示（初始）特征的数量。

表7-1：机器学习模型的复杂度

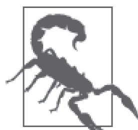
模　型	时　间	空　间
k-均值训练	$O(nkd)$[a]	$O(kd)$
k-均值预测	$O(kd)$	$O(kd)$
LR+ 簇特征训练	$O(n(d+k))$	$O(d+k)$
LR+ 簇特征预测	$O(d+k)$	$O(d+k)$
RBF SVM 训练	$O(n^2d)$	$O(n^2)$
RBF SVM 预测	$O(sd)$	$O(sd)$
GBT 训练	$O(nd\,2^mt)$	$O(nd+2^mt)$
GBT 预测	$O(2^mt)$	$O(2^mt)$
kNN 训练	$O(1)$	$O(nd)$
kNN 预测	$O(nd+k\log n)$	$O(nd)$

a 流式 k-均值可以在 $O(nd(\log k + \log\log n))$ 时间内完成，对于很大的 k，这个速度要比 $O(nkd)$ 快很多。

对于 k-均值算法，训练时间为 $O(nkd)$ 是因为每次迭代都要计算每个数据点和每个中心点（k）之间的 d 维距离。我们乐观地假定迭代次数不是 n 的函数，尽管这不是在所有情况下均成立。在预测时，需要计算新数据点和 k 个中心点之间的距离，所以是 $O(kd)$。存储空间的要求是 $O(kd)$，用来保存 k 个中心点的坐标。

逻辑回归的训练和预测对于数据点数量和特征维度都是线性的。RBF SVM 的训练成本很高，因为需要对输入数据的每对数据点计算核矩阵。与训练过程相比，RBF SVM 的预测成本很低，它对于支持向量的数量 s 和特征维度 d 是线性的。GBT 的训练和预测与数据量和模型大小呈线性关系（t 棵树，每棵树最多 2^m 个叶子节点，其中 m 是树的最大深度）。kNN 的简单实现根本不需要训练时间，因为训练数据本身就是实质上的模型。由此造成的代价体现在预测时间上，预测时的输入必须使用所有初始训练数据进行评价，还需要部分排序以找出 k 个最近邻。

总的来说，k-均值 +LR 是唯一一个对于训练数据的规模（$O(nd)$）和模型大小（$O(kd)$）在训练时间和预测时间上都是线性的组合。它的复杂度与 GBT 最为相似，GBT 的成本与数据点数量、特征维度和模型大小（$O(2^mt)$）呈线性关系。但很难说是 k-均值 +LR 还是 GBT 能得到更小巧的模型，这取决于数据的空间性质。



潜在的数据泄露

那些还记得我们对数据泄露（见 5.2.2 节中的“防止数据泄露”部分）的担心的人或许会问：在 k-均值特征化阶段引入目标变量难道不会导致数据泄露吗？答案是“会的”，但不像分箱计数中那么严重。如果我们使用同样的数据集来学习簇和建立分类模型，那么关于目标的信息就会泄露到输入变量中。造成的结果是，**在训练数据上**的准确度评价可能会过于乐观，但是，在没有用于模型训练的验证集或测试集上进行评价时，这种偏差就会被消除掉。而且，这种泄露不会像分箱计数统计量（见 5.2.2 节）中那么糟糕，因为聚类算法的有损压缩会抽象掉一些目标信息。如果需要特别防止数据泄露，可以分割出一个单独的数据集来完成簇的导出，就像分箱计数中做的那样。

k-均值特征化适合实数型、有界的、能在空间中形成块状密集区域的数值特征。块状区域可以是任意形状，因为我们可以增加簇的数量来近似它们。（与经典聚类方法不同，我们不关心如何找出簇的“真实”数目，而只需覆盖它们。）

k-均值不能处理欧氏距离无效的特征空间，即分布奇特的数值型变量或分类变量。如果特征集合中包括这种变量，那么有以下几种处理方法。

(1) 仅在实数型、有界的数值特征上应用 *k*-均值特征化。

(2) 自定义一种度量方式，用来处理多种数据类型，并使用 *k*-中心点算法。（*k*-中心点是一种与 *k*-均值类似的方法，允许使用任意的度量方式。）

(3) 先将分类变量转换为分箱计数统计量（见 5.2.2 节），再使用 *k*-均值对其进行特征化。

与处理分类变量和时间序列的技术相结合，*k*-均值特征化可以用来处理经常出现在像客户营销和销售分析这种情境下的大批量数据。最后得到的簇可以看作对用户的细分，这在随后的建模阶段是非常有用的特征。

7.5　小结

本章介绍了模型堆叠的概念，使用的是一种非传统的方法：将有监督的 *k*-均值聚类和简单的线性分类器结合起来。*k*-均值通常被用作无监督建模方法，目的是在特征空间中找出数据点的密集簇。但在本章中，可以有选择地为 *k*-均值提供类标签作为输入，这有助于 *k*-均值找到与类别边界更加对齐的簇。

下一章将讨论深度学习，它通过将各层神经网络彼此叠加在一起，将模型堆叠提高到了一个新的水平。ImageNet 大规模视觉识别竞赛近期的两位优胜者使用了 13 层和 22 层的神经网络。他们利用了现有的大量未标记训练图像，从中找寻能得到良好图像特征的像素组合。本章使用的技术分别训练了 *k*-均值特征生成器和线性分类器。但是，也可以对特征生成器和分类器进行联合优化。正如我们将看到的，深度学习的训练过程就是采用的这种技术路线。

7.6　参考文献

Ted Dunning 简直就是数据科学的一本行走的百科全书。他经常在业界活动中发表演讲，他喜欢啤酒和有趣的人。你可以和他把酒言欢，你肯定不会感到失望。

Zheng, Alice. Evaluating Machine Learning Models [M]. Sebastopol, CA: O’Reilly Media, 2015.



第8章

自动特征生成：图像特征提取和深度学习

影像和声音是人类固有的感官输入。我们的大脑天生适合快速发展处理视觉和听觉信号的能力，有些系统甚至在出生之前就可以对刺激做出反应（Eliot, 2000）。另一方面，语言能力则是靠学习得到的，它需要几个月来发展，而完全掌握则需要好几年。很多人的视觉和听觉能力的发展都是自然而然的，但所有人都必须有意地训练自己的大脑来理解和使用语言。

有趣的是，对于机器学习来说，情况则正好相反。我们在文本分析应用方面取得的进展要远远多于图像和音频应用。以搜索问题为例，人们已经享受了多年在信息检索和文本搜索方面的成果，而图像和音频搜索还在走向成熟的途中（然而在过去的 5 年中，深度学习模型取得了突破性发展，这可能预示着在图像和语音分析领域会出现人们期待已久的革命性成果）。

进展中的困难与从图像和音频数据中提取有意义特征的难度直接相关。机器学习模型需要语义上有意义的特征来做出语义上有意义的预测。在文本分析中，尤其是在像英语这样语义上有意义的基本单位（单词）很容易提取的语言中，进展可以非常快速。另一方面，图像和声音是以数字像素或波形来记录的。图像中的单个“原子”是一个像素。在音频数据中，基本单位是对波形密度的一次测量。这些单位包含的语义信息要比文本数据的基本单位（单词）少。因此，与文本相比，图像和音频上的特征提取和特征工程要困难得多。

在过去 20 年中，计算机视觉研究的重点是人工定义的用于提取良好图像特征的流程。有一段时间，像 SIFT 和 HOG（后文中会介绍）这样的图像特征提取器曾经成为了标准。近期深度学习研究的发展扩展了传统机器学习模型的应用范围，它们在基础层中集成了自动特征提取技术。这种扩展实质上是使用人工定义的能自动学习和提取特征的模型代替人工定义的特征图像提取器。人工工作依然存在，只是更加深入地抽象到了建模过程内部。

在本章中，我们从最流行的图像特征提取器开始，然后详细介绍本书中最为复杂的建模机制——用于特征学习的深度学习。

8.1 最简单的图像特征（以及它们因何失效）

从图像中提取出的**正确**特征是什么？答案当然取决于特征的用途。假设我们的任务是图像检索：给定一张图片，我们要从一个图像数据库中找出与其相似的图片。我们需要确定如何表示每张图像，以及如何测量图像之间的区别。我们能只考虑不同颜色在图像中的比例吗？图 8-1 中给出了两张图片，它们具有大致相同的颜色配置，但是含义则大相径庭。一张似乎是蓝天上的白云，另一张则是海上的帆船。因此，仅凭颜色信息可能还不足以标识一幅图像。

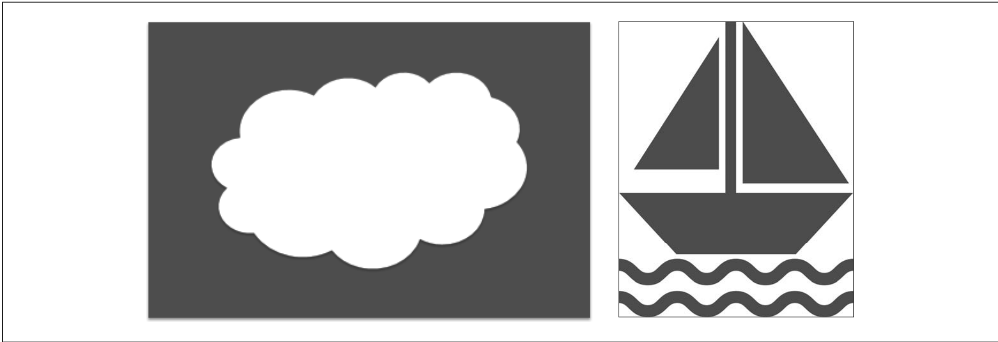


图 8-1：蓝色与白色组成的图片——同样的颜色配置，完全不同的意义

另外一种简单思想是测量图像之间的像素值差异。首先，调整图像，使它们具有同样的宽度和高度。每幅图像都用一个像素值矩阵来表示，这个矩阵可以按行或按列堆叠成一个长向量。每个像素的颜色（如颜色的 RGB 编码）就是图像的一个特征。最后，测量出长像素向量之间的欧氏距离。这种方法确实能够分辨出海上帆船和白云图片，但要作为一种相似度测量，它就过于严格了。白云的形状可以千变万化，但仍旧是云。它可以移到图像的一边，也可以半掩在阴影里。所有这些变化都会增加欧氏距离，但不会改变图像仍旧是云这一事实。

问题在于，单个像素不能携带足够的关于图像的语义信息。因此，对于分析来说，它们不是合适的基本单位。

8.2 人工特征提取：SIFT和HOG

1999 年，计算机视觉研究者找到了一种更好的图像表示方法，它使用的是小块图像的统计量，这种方法称为**尺度不变特征转换**（scale invariant feature transform，SIFT）[Lowe, 1999]。

SIFT 最初是为对象识别任务而开发的。对象识别不仅要正确地标注出图像中包含某种对象，还要确定对象在图像中的位置。对象识别过程包括在各种可能的尺度层次上分析图像，检测能够表示对象存在的有趣的点，提取这些有趣的点的相关特征（在计算机视觉中通常称为**图像描述符**），以及确定对象的姿态。

近年来，SIFT 的使用范围扩大了，不仅仅用于我们感兴趣的点，还可以提取整个图像的特征。SIFT 特征提取过程与另一种技术非常相似，那就是梯度方向直方图（histogram of oriented gradients，HOG）[Dalal and Triggs, 2005]。这两种技术本质上都是计算梯度方向上的直方图，下面进行详细的介绍。

8.2.1 图像梯度

要想比原始像素值做得更好，必须以某种方式将像素“组织”成信息量更大的单位。相邻像素之间的差别通常是非常有用的。像素值一般在对象的边界发生改变，比如一片阴影，一个模式之中，或者一个纹理表面。相邻像素在值上的差异称为**图像梯度**。

计算图像梯度的最简单方法是先沿着图像的横轴（x）和纵轴（y）分别计算像素值的差异，然后将计算结果组合到一个二维向量中。这需要两个一维差分操作，可以用向量掩膜或滤波器方便地表示。掩膜 [1, 0, –1] 既可以计算左邻和右邻之间的差异，也可以计算上邻和下邻之间的差异，这取决于应用掩码的方向。也可以使用二维梯度滤波器，但在这个例子中，一维滤波器就足够了。

要对图像应用滤波器，可以执行一个**卷积**操作：先对滤波器做个翻转，再与一小块图像做内积，然后移动到下一小块。卷积是信号处理中的常用操作，我们使用 * 来表示这种操作：

$$[a\ b\ c] * [1\ 2\ 3] = c*1 + b*2 + a*3$$

像素 (i, j) 的 x 梯度和 y 梯度分别是：

$$g_x(i,j) = [1\ 0\ -1] * [I(i-1,j)\ I(i,j)\ I(i+1,j)] = -1 * I(i-1,j) + 1 * I(i+1,j)$$

$$g_y(i,j) = [1\ 0\ -1] * [I(i,j-1)\ I(i,j)\ I(i,j+1)] = -1 * I(i,j-1) + 1 * I(i,j+1)$$

放在一起，它们就构成了梯度：

$$\nabla I(i,j) = \begin{bmatrix} g_x(i,j) \\ g_y(i,j) \end{bmatrix}$$

向量可以完全由它的方向和大小来表示。梯度大小等于梯度的欧几里得范数$\left(\sqrt{g_x^2+g_y^2}\right)$，它表示像素周围的像素值变化的程度。梯度方向取决于水平和垂直方向上改变的相对大小，它可以这样计算：$\theta = \arctan\left(\frac{g_y}{g_x}\right)$。图 8-2 演示了这些数学概念。

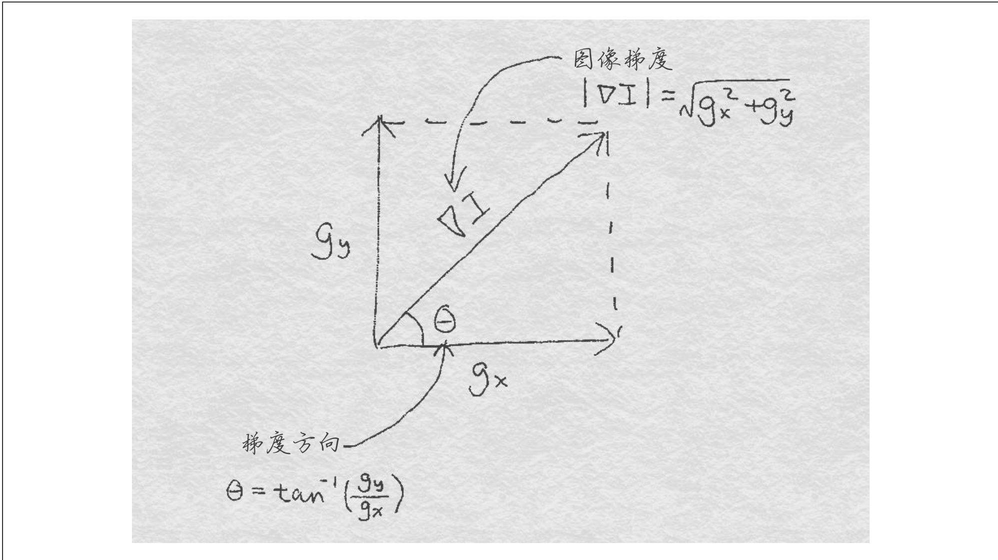


图 8-2：图像梯度定义示意图

图 8-3 给出了几个图像梯度的简单例子，包括垂直梯度和水平梯度。每个例子都是一个 9 像素图像，每个像素都用灰度值进行标记。（数字越小，颜色越深。）中间像素的梯度显示在每个图像下方。左侧图像中有几个水平的色条，颜色只在垂直方向上变化，因此，水平梯度为 0，垂直梯度不为 0。中间图像中有几个垂直的色条，因此，水平梯度不为 0。右侧图像的色条在对角线上，梯度方向也是沿着对角线的。

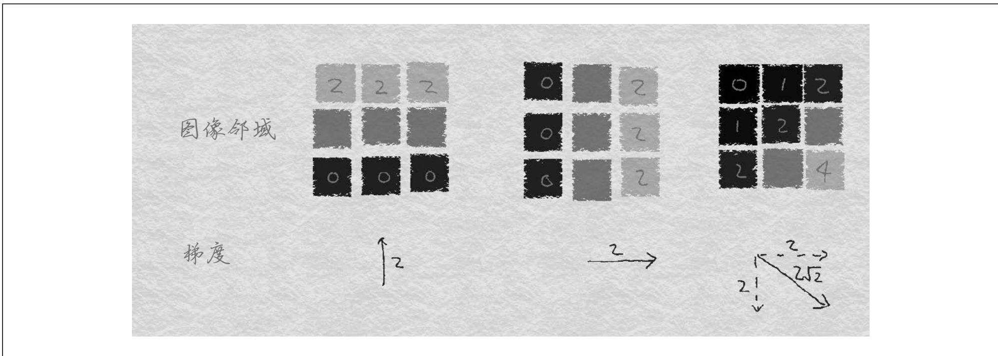


图 8-3：图像梯度的简单例子

这个定义在人造示例上是有效的，但对真实的图像呢？在例 8-1 中，我们使用 `scikit-image` 中的一张猫的图片来检查一下这个问题，图像的水平梯度和垂直梯度显示在图 8-4 中。因为这些梯度是在原始图像的每个像素位置上计算的，所以可以得到两个新的矩阵，每个矩阵都可以可视化为一个图像。

例 8-1　使用 Python 计算简单图像梯度

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> from skimage import data, color

### 加载示例图像，并转换为灰度模式
>>> image = color.rgb2gray(data.chelsea())

### 使用中心化的一维滤波器计算水平梯度。
### 这等价于将每个非边界像素替换为
### 它的左侧相邻像素和右侧相邻像素的差。
### 最左边和最右边上的像素的梯度是0。
>>> gx = np.empty(image.shape, dtype=np.double)
>>> gx[:, 0] = 0
>>> gx[:, -1] = 0
>>> gx[:, 1:-1] = image[:, :-2] - image[:, 2:]

### 以同样的方式计算垂直梯度
>>> gy = np.empty(image.shape, dtype=np.double)
>>> gy[0, :] = 0
>>> gy[-1, :] = 0
>>> gy[1:-1, :] = image[:-2, :] - image[2:, :]

### Matplotlib命令
>>> fig, (ax1, ax2, ax3) = plt.subplots(3, 1,
...                                     figsize=(5, 9),
...                                     sharex=True,
...                                     sharey=True)

>>> ax1.axis('off')
>>> ax1.imshow(image, cmap=plt.cm.gray)
>>> ax1.set_title('Original image')
>>> ax1.set_adjustable('box-forced')

>>> ax2.axis('off')
>>> ax2.imshow(gx, cmap=plt.cm.gray)
>>> ax2.set_title('Horizontal gradients')
>>> ax2.set_adjustable('box-forced')

>>> ax3.axis('off')
>>> ax3.imshow(gy, cmap=plt.cm.gray)
>>> ax3.set_title('Vertical gradients')
>>> ax3.set_adjustable('box-forced')
```

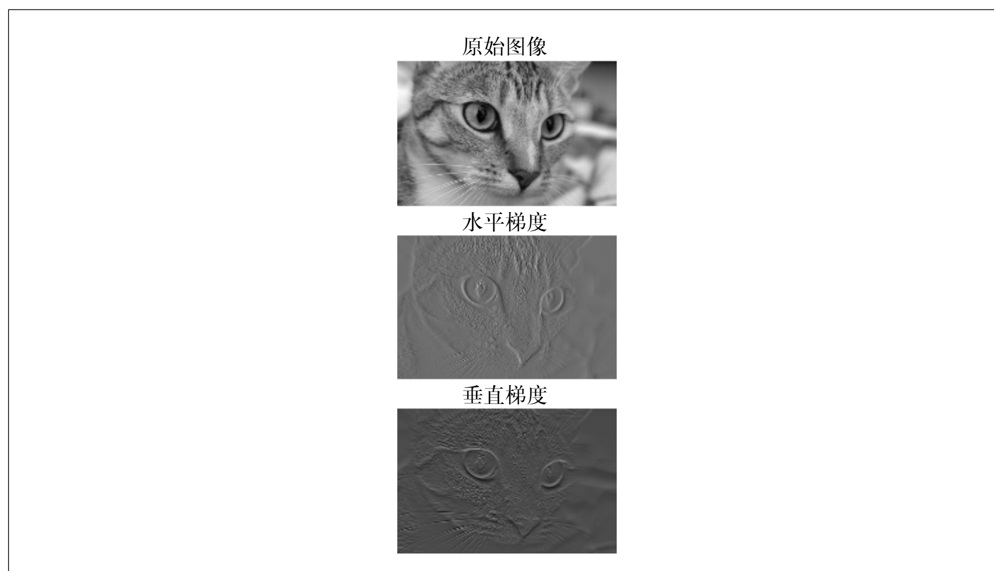


图 8-4：猫图像的梯度

请注意，水平梯度提取出了强烈的**垂直**模式，比如猫眼睛的内部边缘，而垂直梯度则提取出了强烈的**水平**模式，比如胡须和上下眼睑。乍看上去，这可能有点矛盾，但如果仔细思考一下，就会明白确实应该是这样的。水平（x）梯度识别的是水平方向上的改变，一个强烈的垂直模式会在基本相同的 x 位置跨越多个 y 像素，因此，垂直模式会造成像素值在水平方向上的差异。这也是我们的眼睛检测到的结果。

8.2.2　梯度方向直方图

单个的图像梯度可以提取出图像相邻区域中的微小差异，但我们的眼睛看到的则是更大的模式。例如，我们看到的是猫的整条胡须，而不仅仅是一个小的部分。人类视觉系统能识别出一个区域中的连续模式，所以要想总结出邻近区域的图像梯度，还有更多工作要做。

应该将向量描述得多么精确？统计学家会说："看它的分布！"SIFT 和 HOG 都是这样做的。具体说来，它们计算（归一化）梯度向量的直方图作为图像特征。直方图将数据分装到多个箱子中，并计算出每个箱子中数据点的数量，这是一种（未归一化的）经验分布。归一化可以保证所有计数的总和为 1，用数学语言说就是它具有单位 ℓ^1 范数。

图像梯度是个向量，而向量可以通过两个分量表示：方向和大小。所以，我们还需要确定如何设计直方图来表示这两个分量。SIFT 和 HOG 给出了一种表示方法，将图像梯度按照它们的方向角 θ 进行分箱，按照每个梯度的大小进行加权。具体步骤如下。

(1) 将 $0^\circ \sim 360^\circ$ 进行等宽分箱。

(2) 对于邻域中的每个像素，在与其方向角 θ 对应的分箱中添加一个权重 w。w 是该像素的梯度大小与其他相关信息的一个函数。例如，这种信息可以是该像素与小块图像中心点距离的倒数。指导思想是，如果梯度大，权重也应该大，而且靠近图像邻域中心的像素的重要程度要高于那些远离中心的像素。

(3) 对直方图进行归一化。

图 8-5 给出了一个在 4 像素 ×4 像素的图像邻域上的 8 分箱梯度方向直方图示意图。

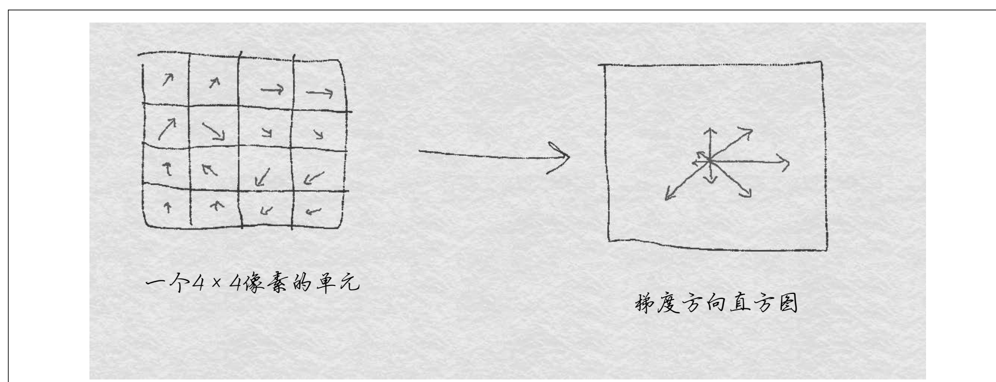


图 8-5：基于 4×4 方块单元像素梯度的 8 分箱梯度方向直方图示意图

当然，在基本梯度方向直方图算法中，有很多设置和选项需要调整。通常来说，正确的设置与要分析的具体图像高度相关。

下面研究一下几个需要确定的问题以及它们对模型的影响。

1. 需要多少个分箱？它们的范围是0°~360°（有符号梯度）还是0°~180°（无符号梯度）？

更多的分箱可以对梯度方向进行粒度更细的量化，由此可以保留关于初始梯度的更多信息。但是，过多的分箱不但没必要，还会导致对训练数据的过拟合。例如，你可能不会根据方向正好在 3° 的猫的胡须来识别图像中的猫。

还有一个问题，分箱范围应该是 0° ~ 360°（这会保留沿着 y 轴的梯度的正负号），还是 0° ~ 180°（这不会保留垂直梯度的正负号）呢？在首次提出 HOG 方法的论文中，作者（Dalal and Triggs, 2005）通过实验确定了范围为 0° ~ 180° 的 9 分箱方式是最好的，而 SIFT 论文（Lower, 2004）则推荐 0° ~ 360° 的 8 分箱方式。

2. 使用哪种权重函数？

HOG 论文比较了多种梯度大小加权方式：向量本身的长度、长度的平方或平方根、二值化，或在高端或低端进行裁剪。不做任何处理的普通长度在作者的实验中效果最好。

SIFT 使用的也是梯度本身的长度。此外，它还想避免由于图像窗口位置的微小变化而导致的特征描述符中的突变，因此，它使用一种从窗口中心进行测量的高斯距离函数，对来

自邻域边缘的梯度赋予较低的权重。换句话说，即用梯度大小乘以$\frac{1}{2\pi\sigma^2}e^{-\|p-p_0\|^2/2\sigma^2}$，其中 p 是生成梯度的像素位置，p_0 是图像邻域的中心点，高斯宽度 σ 设为邻域半径的一半。

SIFT 还试图避免由于单个图像梯度方向的微小变化而导致的在梯度方向直方图中发生的较大变化。因此，它使用了一种插值技巧，将权重从一个单独梯度扩展到相邻方向的分箱。特别地，根分箱（梯度所在的分箱）获得的投票是 1 乘以加权的梯度长度。每个相邻分箱获得的投票是 $1-d$，其中 d 是直方图分箱单位与根分箱之间的差。

总的来说，SIFT 中一个图像梯度的投票是：

$$w_{(\nabla p,b)} = w_b \sigma \left\|\nabla_p\right\|$$

其中∇_p是分箱 b 中像素 p 的梯度，w_b 是 b 的插值权重，σ 是从 p 到中心点的高斯距离。

3. 如何定义邻域？邻域如何覆盖图像？

HOG 和 SIFT 都采用了图像邻域的一种双层表示法：首先将邻接的像素组成单元，然后再将邻接的单元组织成块。先在每个单元上计算方向直方图，再将单元直方图向量连接起来，构成整个块的最终特征描述符。

SIFT 使用 16 像素 × 16 像素的单元，组织成 8 个方向上的分箱，然后按照 4 × 4 单元的块进行分组，这样图像邻域就有 4 × 4 × 8=128 个特征。

HOG 论文试验了矩形和圆形的单元和块。矩形单元称为 R-HOG 块。能找到的最优 R-HOG 设置是 8 像素 × 8 像素的 9 方向分箱，再分组成为 2 × 2 单元的块。圆形单元称为 C-HOG 块，具体设置是可变的，由中心单元的半径、单元是否按放射状划分、外部单元的宽度等因素确定。

不论邻域如何组织，它们通常通过重叠来形成整个图像的特征向量。换句话说，单元和块沿着水平方向和垂直方向在图像中平移，但每次只移动几个像素，以覆盖整个图像。

邻域结构的主要内容是多层次的组织形式和在图像中平移的重叠窗口。在深度学习网络的设计中，也使用了同样的模式。

4. 应该使用哪种类型的归一化？

归一化可以使特征描述符变得均衡，并且大小是可比较的。它是第 4 章中讨论的缩放的同义词。我们已经发现，在文本特征上的（tf-idf 形式的）特征缩放对于分类准确度没有很大的影响。图像特征则完全不同，它们对自然图像中光照和对比度的变化非常敏感。举个例子，我们考虑两种情况下的苹果图像，一种位于强烈的点光源下，另一种则位于从窗户透过来的柔和散射光线之下。尽管图像中的对象都是相同的，但图像梯度的大小会有非常大的差别。正因为这个原因，一般情况下，计算机视觉中的图像特征化首先要进行全局颜色归一化，目的就是消除照明和对比度上的差异。对于 SIFT 和 HOG，事实证明只要对特征

进行了归一化，这种预处理就不是必要的了。

SIFT 遵循的是一种归一化—阈值—归一化模式。首先，将块特征向量归一化为单位长度（ℓ^2 归一化）。然后，将特征裁剪到最大值，以消除极端的照明效果，比如来自照相机的颜色饱和度。最后，将裁剪后的特征再归一化为单位长度。

HOG 论文试验了多种不同的归一化方法，包括 ℓ^2 范数和 ℓ^1 范数，以及 SIFT 论文中使用的归一化—阈值—归一化模式。作者发现，纯 ℓ^1 归一化的可靠性比其他方法略差一些，其他方法的效果则没有太大区别。

8.2.3　SIFT体系

SIFT 流程需要相当多的步骤。HOG 稍微简单一些，但同样需要很多基本步骤，比如创建梯度直方图和归一化。图 8-6 是 SIFT 体系的示意图。从原始图像的一个感兴趣区域开始，首先将该区域划分为网格。然后，将每个网格单元进一步划分为子网格。每个子网格元素都包含大量的像素，而且每个像素都生成一个梯度。每个子网格元素都生成一个加权梯度估计，其中的权重是精心选择的，以使子网格元素外部的梯度也有所贡献。然后，这些梯度估计被聚合为子网格的方向直方图，其中的梯度可以具有前面描述过的加权投票。此后，每个子网格的方向直方图被连接成为整个网格的长梯度方向直方图。（如果网格被分成 2×2 个子网格，那么就会有 4 个梯度方向直方图被连接成 1 个。）这就是网格的特征向量，随后它进入归一化—阈值—归一化流程。首先，向量被归一化成具有单位范数的向量。然后，各个值被剪裁到一个最大阈值。最后，对阈值剪裁后的向量再进行归一化。这样，就得到了这块图像区域的最终 SIFT 特征描述符。

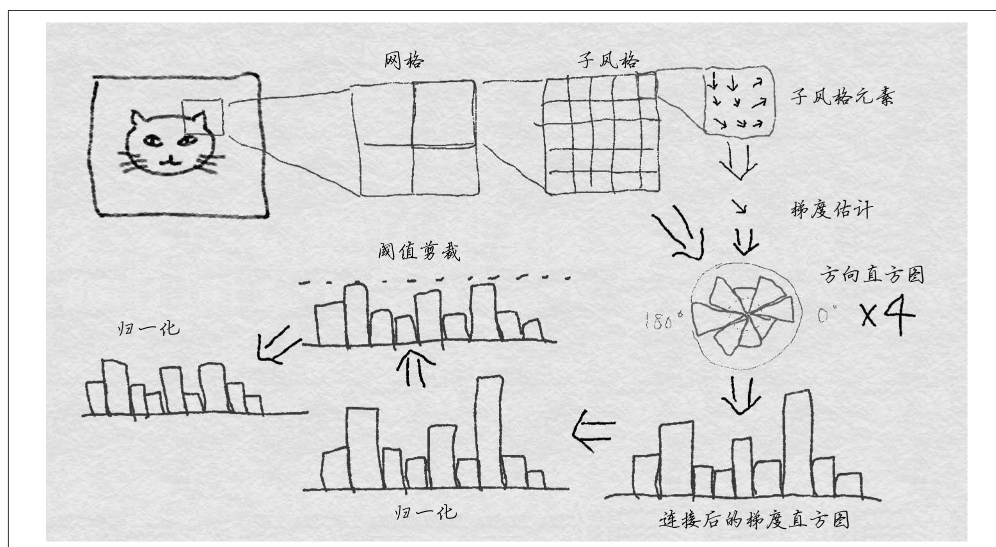


图 8-6：SIFT 体系——原始图像中一块感兴趣区域的特征向量生成步骤

8.3 通过深度神经网络学习图像特征

要定义良好的图像特征，SIFT 和 HOG 还有很长一段路要走。然而，计算机视觉领域的最新成果则来自于另外一个非常不同的方向：深度神经网络模型。这一突破发生在 2012 年的 ImageNet 大规模视觉识别竞赛（ILSVRC）中，当时来自多伦多大学的一群研究者几乎将前一年优胜者的错误率降低了一半。他们称其所使用的方法为“深度学习”，以强调这种方法不同于以前的神经网络模型，而是包括很多叠加在一起的神经网络层和转换层的最新一代模型。ILSVRC 2012 的优胜模型——后来被称为 AlexNET，以它的首席发明者命名——有 13 层（Krizhevsky 等，2012），ILSVRC 2014 的获胜者 GoogleNet 有 22 层（Szegedy 等，2014）。

从表面上看，堆叠神经网络的机制似乎与 SIFT 和 HOG 的图像梯度直方图相去甚远。但从 AlexNet 的可视化可以看出，它的最初几层本质上就是计算边缘梯度和其他一些简单模式，与 SIFT 和 HOG 非常相似。随后的几层将局部模式组合成更全局化的模式。最终结果是一个特征提取器，比以前的提取器要强大许多。

堆叠神经网络层（或任意其他分类模型）这种模式并不是什么新的思想，但训练这种复杂模型需要大量的数据和计算能力，这是近期才具备的。ImageNet 数据集中有 120 万张标记好的图像，类别有 1000 个。现代 GPU 大大加快了矩阵 - 向量计算，很多机器学习模型（包括神经网络）的内核就需要这种计算。深度学习方法的成功就依赖于可使用大量数据和大量 GPU 时间。

深度学习体系由几种不同类型的层组成。例如，AlexNet 包括全连接层、卷积响应归一化层和最大池化层。下面依次进行介绍。

8.3.1 全连接层

所有神经网络的核心都是输入的线性函数。我们在第 4 章中遇到的逻辑回归就是神经网络的一个例子。全连接神经网络就是所有输入特征的线性函数集合。回忆一下，线性函数可以写作输入特征向量和权重向量的内积，再加上一个可能的常数项。线性函数集合可以用矩阵和向量的乘积来表示，将权重向量变成权重矩阵（$\boldsymbol{W}$）即可。

全连接层的数学定义为：

$$z = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

其中 $\boldsymbol{W}$ 的每一行都是一个权重向量，可以将整个输入向量 $\boldsymbol{x}$ 映射为 z 中的一个输出。$\boldsymbol{b}$ 是一个由标量组成的向量，表示每个神经元固定的偏移（或偏差）。

之所以称为全连接层，是因为每个输入都可以用于每个输出。在数学上，这意味着对矩阵 $\boldsymbol{W}$ 中的值没有限制。（正如我们即将看到的，卷积层对每个输出只能使用输入的一个小型

子集。）如果用图表示的话，全连接神经网络可以表示为一个完全二分图，其中每个输入节点都连接到每个输出节点（见图 8-7）。

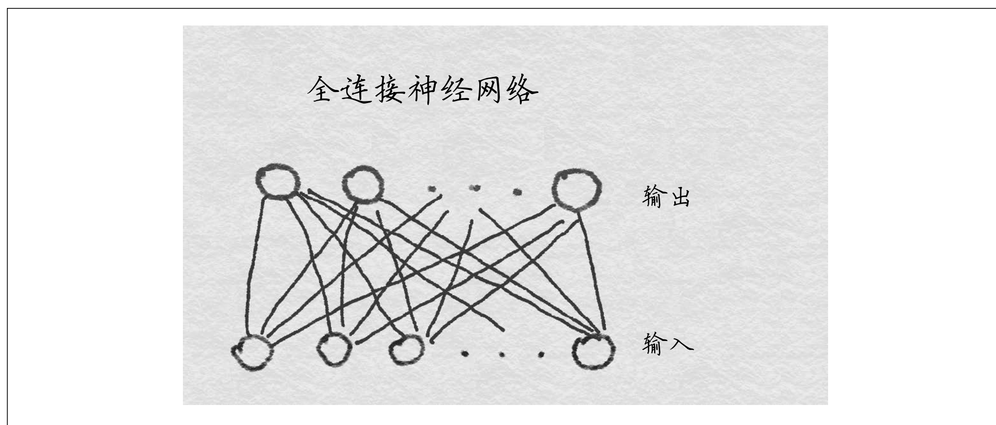


图 8-7：全连接神经网络的图表示

全连接层包含最大可能数目的参数（输入数 × 输出数），因此它非常昂贵。如此密集的连接可以让网络探测包含所有输入的全局模式。由于这个原因，AlexNet 的最后两层是全连接层。全连接层的输出彼此之间还是独立的，依输入情况而定。

8.3.2 卷积层

与全连接层不同，卷积层仅使用输入的一个子集来生成输出。转换过程在输入中"移动"，每次使用若干特征生成输出。为简单起见，我们可以对不同输入集合使用同样的权重，而不用为每个输入集合学习新权重。

在数学上，卷积算子接受两个函数作为输入，生成一个函数作为输出。它要先翻转一个输入函数，再将它沿着另一个函数移动，并输出在每个点上递增的曲线下的面积：

$$(f*g)(t)\int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau = \int_{-\infty}^{\infty} g(\tau)f(t-\tau)d\tau$$

计算一条曲线下总面积的方法是求它的积分。这个算子对于输入是对称的，也就是说，翻转第一个函数还是第二个函数都没有关系，输出是一样的。

在介绍图像梯度时（8.2.1 节），我们已经见过了一个简单的卷积示例。但卷积的数学定义还是显得有些复杂，这自有其原因。使用信号处理中的例子来解释卷积背后的思想是最容易的。

假设有一个小黑盒。为了知道这个黑盒的用处，我们向它传递一个单位的刺激信号，然后在一张纸上记录输出结果，直到再没有对初始刺激的反应为止。最后得到的随时间变化的

函数就是**响应函数**，我们称它为 $g(t)$。

再假设我们有某种疯狂的信号 $f(t)$，我们把它持续地输入到黑盒中。在时刻 $t = 0$，$f(0)$ 与黑盒产生作用，生成了 $f(0)$ 乘以 $g(0)$。在时刻 $t = 1$，$f(1)$ 进入黑盒，生成了 $f(1)$ 乘以 $g(0)$。与此同时，黑盒继续对前面的信号 $f(0)$ 产生响应，这时就要乘以 $g(1)$。所以，时刻 $t = 1$ 的总输出是 $(f(0)*g(1)) + (f(1)*g(0))$。在时刻 $t = 2$，随着 $f(2)$ 的加入，$f(0)$ 和 $f(1)$ 也会继续产生响应，情况就更复杂了。时刻 $t = 2$ 的总输出是 $(f(0)*g(2)) + (f(1)*g(1)) + (f(2)*g(0))$。通过这种方式，$\tau = 0$ 总是与当前输入到黑盒中的信号发生作用，响应函数的尾部与之前的输入信号发生作用，响应函数实际上发生了时间的翻转。

图 8-8 展示了每个时间点上的相互作用。（注意，为了描述方便，我们把时间离散化了。实际上，时间是连续的，所以相加实际上是积分。）如果要计算在一个特定时间点上的卷积值，你需要将两个重叠的信号相乘，再累加起来。

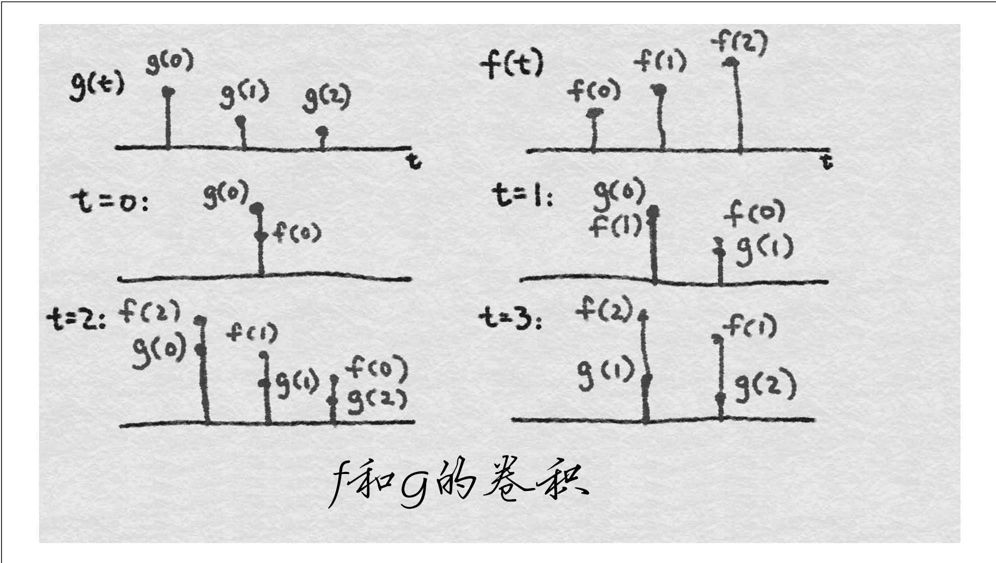


图 8-8：两个离散信号 *f* 和 *g* 的卷积

这种黑盒被称为线性系统，因为它只进行标量相乘和累加。卷积算子清楚地捕获了线性系统的效果。



卷积背后的思想

卷积算子捕获线性系统的效果，而线性系统将接收的信号与它的响应函数相乘，求出当前响应在所有过去输入上的总和。

在我们的例子中，$g(t)$ 表示响应函数，$f(t)$ 表示输入函数。但因为卷积是对称的，所以哪个

是响应函数、哪个是输入函数并不重要，输出就是二者的组合。$g(t)$ 也称为**滤波器**。[1]

图像是二维信号，所以我们需要一个二维滤波器。通过对两个变量进行积分，二维**卷积滤波器**扩展了一维滤波器：

$$(f*g)[i,j]=\sum_{u=0}^{m}\sum_{v=0}^{n}f[u,v]g[i-u,j-v]$$

因为数字图像中是离散的像素，所以卷积积分也变成了离散求和。而且，因为像素数目是有限的，所以滤波器函数只需要有限数量的元素。在图像处理中，二维卷积滤波器也称为**核**或**掩膜**。

当对一张图像应用卷积滤波器时，不一定要定义一个能覆盖整个图像的大型滤波器。相反，只要能覆盖几个像素的小型滤波器就可以了，可以沿着图像的水平方向或垂直方向移动并应用同一个滤波器（见图 8-9）。

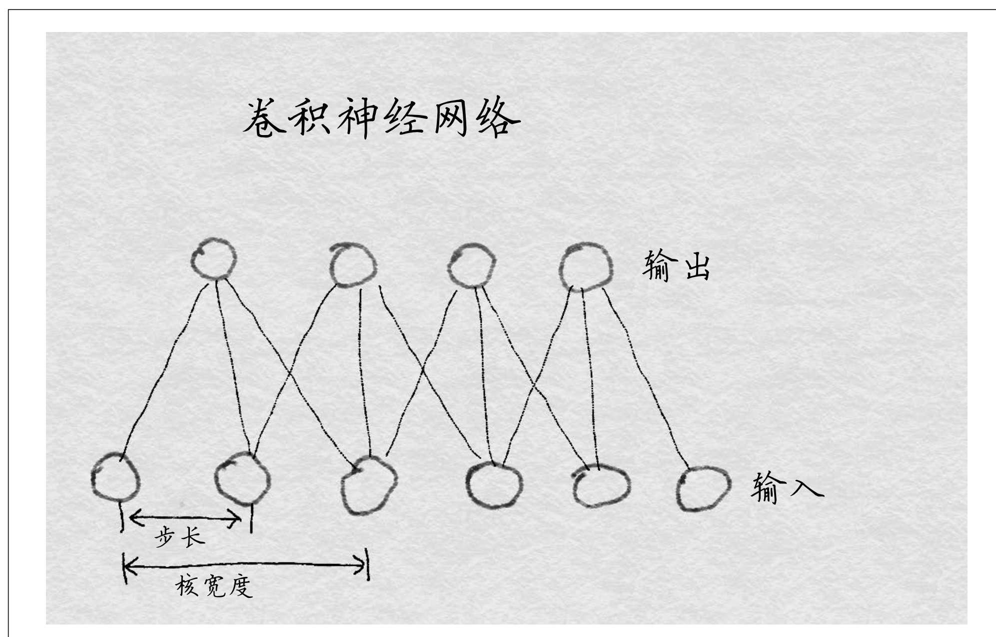


图 8-9：一维卷积神经网络的结构

因为可以在图像上使用同一个滤波器，所以只需定义一小组参数。需要权衡的是，这种滤波器每次只能在一个小的像素邻域内提取信息。换句话说，卷积神经网络识别的是局部模式，而非全局模式。

注 1：技术上，滤波器是一种转换，用来消除傅里叶波谱中的特定部分，但现在人们经常将“滤波器”作为一种通用名称。

卷积滤波器示例

在这个例子中，我们对一张图像应用高斯滤波器。高斯函数在 0 周围形成了一个平滑对称的山丘形状。这种滤波器可以生成附近函数值的加权平均。当应用在图像上时，它的效果是模糊附近的像素值。二维高斯滤波器定义如下：

$$G(x,y)=\frac{1}{2\pi\sigma}\mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}$$

其中 σ 是高斯函数的标准差，它可以控制“山丘”的宽度。

在例 8-2 中，我们先创建一个二维高斯滤波器，然后用它对小猫图像求卷积，生成一张模糊的小猫图像（见图 8-10）。请注意，这不是最精确的计算高斯滤波器的方法，但它是最容易理解的。更好的实现方法是在每个离散点（而不是简单的点估计）上计算加权平均。

例 8-2　在图像上应用简单高斯滤波器

```
>>> import numpy as np

# 先创建5×5的X，Y网格矩阵，用来计算高斯滤波器
>>> ind = [-1., -0.5, 0., 0.5, 1.]
>>> X,Y = np.meshgrid(ind, ind)
>>> X
array([[-1. , -0.5,  0. ,  0.5,  1. ],
       [-1. , -0.5,  0. ,  0.5,  1. ],
       [-1. , -0.5,  0. ,  0.5,  1. ],
       [-1. , -0.5,  0. ,  0.5,  1. ],
       [-1. , -0.5,  0. ,  0.5,  1. ]])

# G是个简单、未归一化的高斯核，其中(0, 0)处的值为1.0
>>> G = np.exp(-(np.multiply(X,X) + np.multiply(Y,Y))/2)
>>> G
array([[ 0.36787944,  0.53526143,  0.60653066,  0.53526143,  0.36787944],
       [ 0.53526143,  0.77880078,  0.8824969 ,  0.77880078,  0.53526143],
       [ 0.60653066,  0.8824969 ,  1.        ,  0.8824969 ,  0.60653066],
       [ 0.53526143,  0.77880078,  0.8824969 ,  0.77880078,  0.53526143],
       [ 0.36787944,  0.53526143,  0.60653066,  0.53526143,  0.36787944]])

>>> from skimage import data, color
>>> cat = color.rgb2gray(data.chelsea())

>>> from scipy import signal
>>> blurred_cat = signal.convolve2d(cat, G, mode='valid')

>>> import matplotlib.pyplot as plt
>>> fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,4),
...                                sharex=True, sharey=True)

>>> ax1.axis('off')
>>> ax1.imshow(cat, cmap=plt.cm.gray)
>>> ax1.set_title('Input image')
>>> ax1.set_adjustable('box-forced')
```

```
>>> ax2.axis('off')
>>> ax2.imshow(blurred_cat, cmap=plt.cm.gray)
>>> ax2.set_title('After convolving with a Gaussian filter')
>>> ax2.set_adjustable('box-forced')
```



图 8-10：应用二维高斯滤波器前后的小猫图片

AlexNet 中的卷积层是三维的，也就是说它们处理的是来自于前一层的三维像素（表示三维图像空间的数组值）。第一个卷积神经网络接受原始 RGB 图像并学习出卷积滤波器，可用于在所有三个颜色通道上的局部图像邻域。随后的几层接受输入的三位像素，沿着空间和核维度进行处理。参考图 8-14 获得更多详细信息。

8.3.3　ReLU变换

神经网络的输出通常被传递到另一种非线性变换中，这种非线性变换又称为**激活函数**。常用的激活函数包括 tanh 函数（一个位于 −1 和 1 之间的平滑非线性函数）、sigmoid 函数（一个位于 0 和 1 之间的平滑非线性函数，在 4.3 节中介绍过），以及所谓的**线性整流函数**（ReLU）。ReLU 是线性函数的一个简单变种，它的负数部分都被归零。也就是说，它裁剪掉了负数部分，保留了无界的正数部分。ReLU 的取值范围是 0 到 ∞。

常用激活函数

ReLU 是一个负数部分归零的线性函数：

$$\mathrm{ReLU}(x) = \max(0, x)$$

tanh 函数是一个平滑地从 −1 增长到 1 的三角函数：

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}}$$

sigmoid 函数平滑地从 0 增长到 1：

$$\mathrm{sigmoid}(x) = \frac{1}{1 + \mathrm{e}^{-x}}$$

图 8-11 给出了这 3 个函数的示意图。

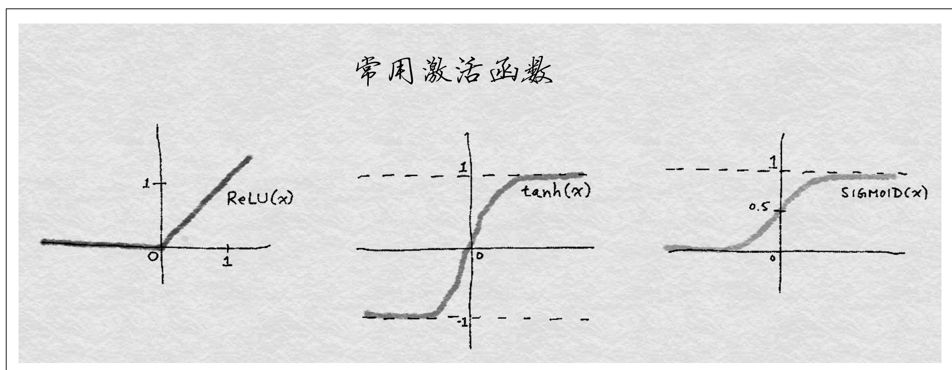


图 8-11：3 个常用激活函数的示意图：ReLU、tanh 和 sigmoid

ReLU 变换对非负函数没有效果，如原始图像或高斯滤波器。然而，训练后的神经网络，不管是全连接类型还是卷积类型，都可能输出负值。AlexNet 使用 ReLU 变换而不是其他变换，在训练时可以更快地收敛（Krizhevsky 等，2012）。它在所有卷积层和全连接层都使用 ReLU。

8.3.4 响应归一化层

经过了第 4 章和本章前面的讨论，归一化这个概念我们已经耳熟能详。归一化用单个输出除以全部响应的一个函数。所以，理解归一化的另一种方式是，它在邻居之间建立了一种竞争关系，因为这样每个输出的强度都要相对于它的邻居来测量（见图 8-12）。AlexNet 在不同核的每个位置上对输出进行归一化。

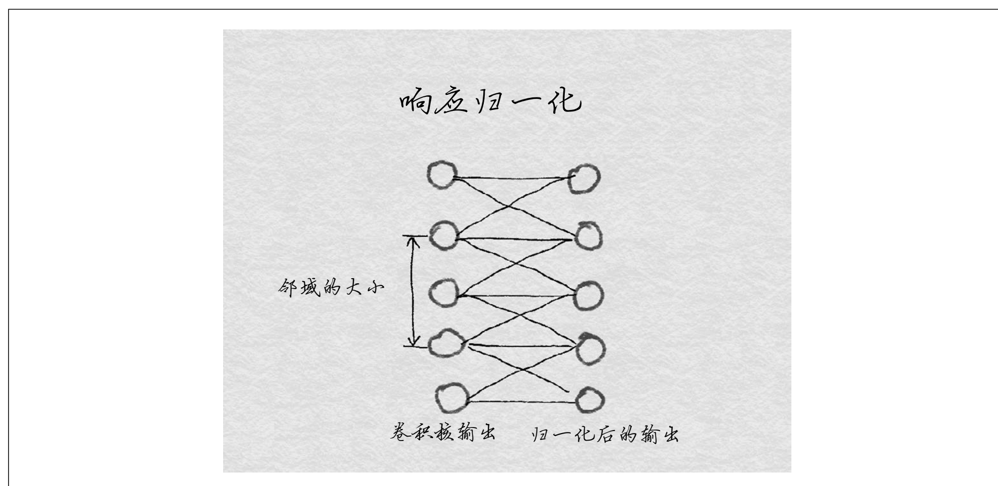


图 8-12：来自上一层卷积核输出的响应归一化结构，归一化常数是根据上一层的邻域计算得出的

局部响应归一化在相邻核之间引起竞争

顾名思义，局部响应归一化将某个值除以它的邻居组合，公式如下：

$$y_k = x_k \Big/ \left(c + \alpha \sum\nolimits_{\ell \in \text{neighborhood of } k} x_\ell^2\right)^\beta$$

其中，x_k 是第 k 个核的输出，y_k 相对于邻域中其他核的归一化响应。对于每个输出位置，归一化是分别进行的；也就是说，对每个输出位置 (i, j)，我们都通过附近的卷积核输出进行归一化。注意，这不同于通过图像邻域或输出位置进行的归一化。核邻域的大小 c、α、β 都是通过图像验证集调优的超参数。

8.3.5 池化层

池化层将多个输入组合成一个输出。当卷积滤波器在图像上移动时，每处理一个邻域都会生成一个输出。池化强制一个局部图像邻域生成一个值，而不是多个值。这可以减少深度学习网络中间层的输出数量，从而有效降低网络在训练数据时出现过拟合的概率。

有多种方法可以池化输入：求平均值、求和（或计算广义的范数）或取最大值。池化可以沿着图像或中间输出层移动进行。AlexNet 使用重叠的最大值池化，在图像中以两个像素（或输出）的步长进行移动，在 3 个邻居之间进行池化，见图 8-13。

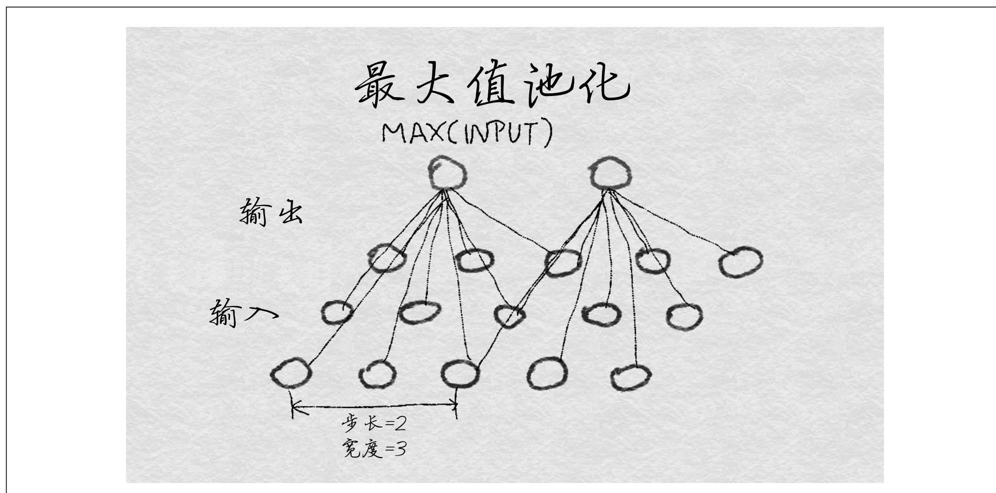


图 8-13：最大值池化使用非线性下采样在每个子区域输出非重叠矩形的最大数量

8.3.6 AlexNet的结构

总体来说，AlexNet 包括 5 个卷积层、2 个响应归一化层、3 个最大值池化层和 2 个全连接层。加上最终的分类输出层，模型中一共有 13 个神经网络层，形成了 8 个层组，详见图 8-14。

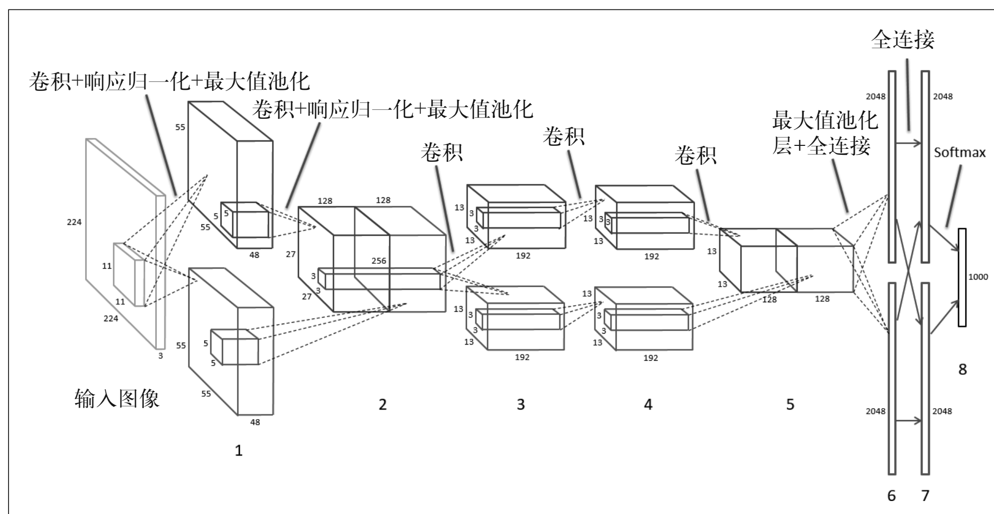


图 8-14：AlexNet 体系结构图——不同的灰度（如果你用全彩模式看这张图，就是品红和蓝色）分别表示驻留在 GPU 1 和 GPU 2 上的层

输入图像首先被缩放为 256 像素 × 256 像素，实际上被剪裁为 224 像素 × 224 像素大小，带有 3 个颜色通道。前两个卷积层后面都分别跟有一个响应归一化层和一个最大值池化层，最后一个卷积层后面跟有一个最大值池化层。原始论文将训练数据和计算分布在两个 GPU 上，层之间的通信大多限制在同一 GPU 中，例外是层组 2 和层组 3 之间以及层组 5 之后。在那些边界点上，下一层接受一个来自于上一层的三维像素作为核的输入，上一层可以位于任意一个 GPU 上。每个中间层后面都要跟随 ReLU 变换。

图 8-15 展示了卷积 + 响应归一化 + 最大值池化的详细视图。注意，归一化常数是在核之间计算的，而池化则发生在图像区域之间。而且，池化会降低层的维度。

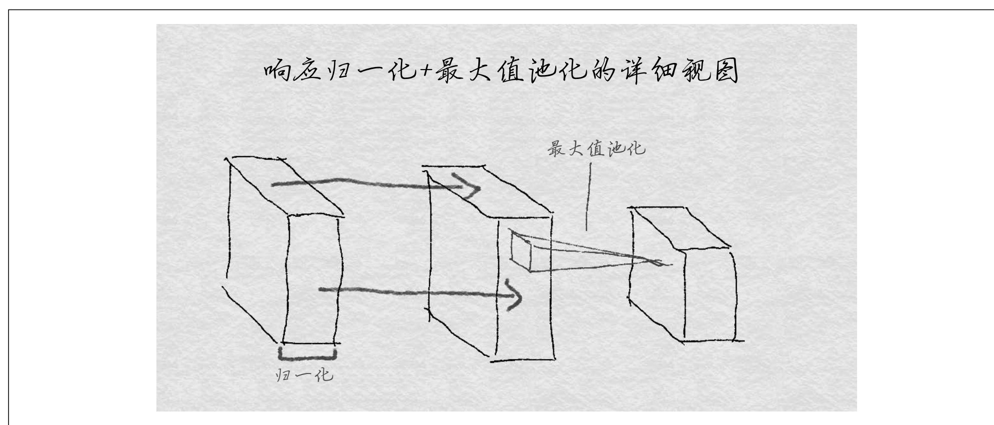


图 8-15：卷积 + 响应归一化 + 最大值池化的详细视图

注意，AlexNet 的体系结构令我们想起了 SIFT/HOG 特征提取器的梯度直方图—归一化—阈值—归一化体系（见图 8-6），只是它具有更多的层。（这就是“深度学习”中“深度”的由来。）然而，与 SIFT/HOG 不同的是，卷积核与全连接权重是从数据中学习出来的，而不是预先定义的。而且，SIFT 中的归一化步骤是在整个图像区域的特征向量之间进行的，而 AlexNet 中的响应归一化层是在卷积核之间执行归一化的。

从一个较高的角度来看，这个模型首先从局部图像邻域提取模式。每个后续的层都建立在前面的层的输出之上，有效地覆盖了初始图像的连续大片区域。因此，即使前 5 个卷积层都只有较小的核宽度，后面的层还是能构造出更为全局化的模式，最后的全连接层则是最为全局化的。

尽管模式的概念非常清晰，但要将每个层提取出的实际模式可视化也是非常困难的。图 8-16 和图 8-17 给出了由模型中学习出的卷积核的前两层的可视化。第一层包括不同方向上的灰度边缘和纹理探测器、色团和纹理。第二层似乎包括各种平滑模式探测器。

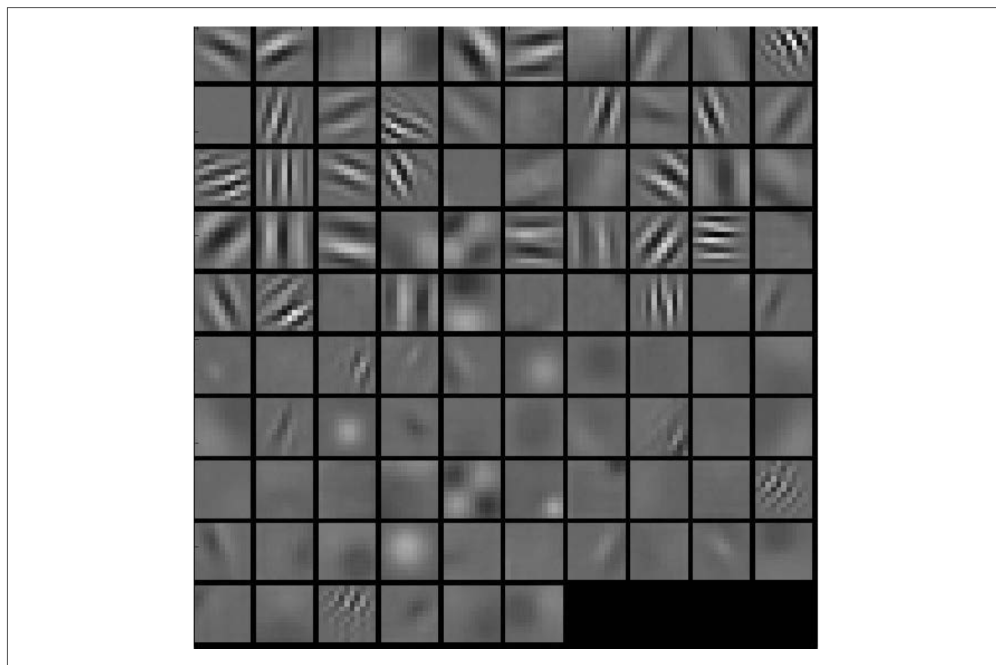


图 8-16：训练后 AlexNet 中第一层卷积核的可视化：一半核是在 GPU 1 上学习的，似乎是检测不同方向上的灰度边缘和纹理的；另一半在 GPU 2 上训练，主要检测色团和模式

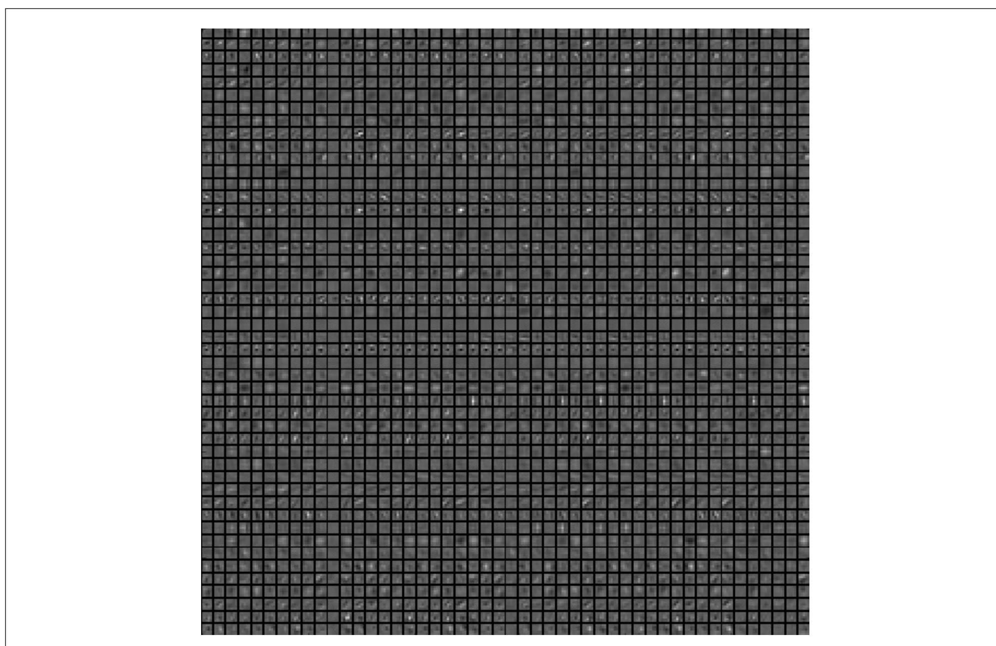


图 8-17：训练后 AlexNet 中第二层卷积核的可视化

尽管这个领域取得了重大进展，但图像特征化还是更像一门艺术，而不是科学。十年前，研究者们使用图像梯度、边缘检测、方向、空间线索、平滑和归一化等技术组合进行手工特征提取。现在，深度学习体系建立的模型封装了大致相同的技术，但参数是从训练图像中自动学习的。深度学习的神奇魔力就隐藏在模型更深处的一种抽象之中，等待我们去探索！

8.4 小结

在接近尾声的时候，我们可以凭借从本章获得的直觉更好地理解：为什么多数简单直观的图像特征对于执行像图像分类这样的任务并不总是最好的。更重要的事情不是将每个像素表示成原子单位，而是考虑像素与其邻近像素之间的关系。SIFT 和 HOG 本来是为其他任务而开发的，但我们可以修改一下这些技术，通过分析邻域中的梯度更好地从整个图像中提取特征。

近年来的一次技术飞跃是在计算机视觉领域使用深度神经网络，目的是使图像的特征提取得到更好的发展。我们需要记住的重点是，深度学习将很多神经网络层和转换层彼此堆叠在一起。在分别检查时，其中有些层可以提取出相似的特征，这些特征可以认为是人类视觉的基础组成部分：定义线条、梯度、颜色图。

8.5 参考文献

CS231n: Convolutional Neural Networks for Visual Recognition [EB/OL]. http://cs231n.github.io/convolutional-networks/.

Dalal, Navneet, and Bill Triggs. Histograms of Oriented Gradients for Human Detection [C]. Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2005): 886–893.

Eliot, Lise. What's Going On in There? How the Brain and Mind Develop in the First Five Years of Life [M]. New York: Bantam Books, 2000.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton. ImageNet Classification with Deep Convolutional Neural Networks [G]. Advances in Neural Information Processing Systems 25 (2012): 1097–1105.

Lowe, David G. Object Recognition from Local Scale-Invariant Features [C]. Proceedings of the International Conference on Computer Vision (1999): 1150–1157.

Lowe, David G. Distinctive Image Features from Scale-Invariant Keypoints [J]. International Journal of Computer Vision 60:2 (2004): 91–110.

Malisiewicz, Tomasz. From Feature Descriptors to Deep Learning: 20 Years of Computer Vision [EB/OL]. Tombone's Computer Vision Blog, January 20, 2015. http://www.computervisionblog.com/2015/01/from-feature-descriptors-to-deep.html.

Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions [C]. Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (2015): 1–9.

Zeiler, Matthew D., and Rob Fergus. Visualizing and Understanding Convolutional Networks [C]. Proceedings of the 13th European Conference on Computer Vision (2014):818–833.

第 9 章

回到特征：建立学术论文推荐器

“在数学中，你不是去理解事情，你只是习惯它们。”

——约翰·冯·诺依曼

当第一次看到图 1-1 中从数据到结果的路径时，很可能会无所适从。纵贯本书，我们的重点在于介绍特征工程的基本原则，我们使用的是玩具模型和简单明了的数据集，这些例子是有意设计成有说明性和启发性的。

机器学习的例子通常展示最理想的情况和最佳结果，这掩盖了本书中描述的路径中的艰辛。既然基础已经打好，我们就离开模拟数据的简单世界，投入到使用真实的、结构化数据集的特征工程中。在前进的每个阶段中，我们都会研究如何从原始数据生成特征，如何进行特征转换，以及特征工程中需要何种权衡取舍。

先说一下，这个综合示例的目标不是为数据集建立最好的模型，而是演示一下本书中几种技术的实际应用，以及如何更加深入地研究一下各种技术是否可以为建模过程提供价值。

9.1 基于项目的协同过滤

我们的任务是使用微软学术图谱的一个子样本为学术论文建立一个推荐器。这个推荐器对于那些需要搜索论文引用但还没有发现 Google Scholar 的人来说是非常方便的。下面是关于这个数据集的一些统计量。

微软学术图谱数据集

- 它包含 166 192 182 篇论文，可经由 Open Academic Graph 获取，只能用于研究目的。
- 完整数据集的大小是 104GB。
- 每条观测有 18 个变量用以标识论文，包括论文题目、论文摘要、作者姓名、关键字和研究领域。

这个数据集被设计成易于使用数据库存储和读取。对于机器学习模型来说，它不够整洁，需要做一些基本的数据整理。有些教师喜欢省略这个步骤，让学生直接建模并得到结果，以此来提高他们的兴趣。我们可不这么做，一切都从头开始。

第一步是将一些变量整理为正确的形式，建立一个基于项目的协同过滤器，看看能否快速有效地找到那些非常相似的论文。



基于项目的协同过滤之起源

这种方法最初是由 Amazon 公司开发的，作为基于用户的商品推荐算法的一种改进。Sarawar 等人详细介绍了将推荐算法从基于用户切换到基于项目的过程中的困难和收获（Sarawar 等，2001）。

基于项目的协同过滤方法根据项目之间的相似程度来提供推荐。这项工作分为两个阶段：首先找出项目之间的相似度评分，然后对所有评分进行排序，找到前 N 个相似项目作为推荐。

建立基于项目的推荐器

基于项目的推荐器完成以下三项任务。

(1) 生成关于“事物”或项目的信息。
(2) 对所有项目进行评分，找出与某个项目“相似”的其他项目。
(3) 返回评分排序 + 项目。

9.2 第一关：数据导入、清理和特征解析

与所有优秀的科学实验一样，我们从一个假设开始。在这个例子中，我们假定那些大约在同一时间而且在同一研究领域发表的论文对用户是最有用的。我们使用一种简单的方法从完整数据集的一个子样本中解析出这些字段。在生成了简单的稀疏数组后，我们在整个项目数组上运行基于项目的协同过滤器，看看能否得到满意的结果。

基于项目的协同过滤器使用相似度评分来比较项目。在这个例子中，余弦相似度可以在两个非零向量之间提供合理的比较。下面的例子使用的就是余弦距离，它是余弦相似度在正空间中的补集，即：

$$D_c(A, B) = 1 - S_c(A, B)$$

其中 D_c 是余弦距离，S_c 是余弦相似度。

学术论文推荐器：简单方法

第一步就是导入和检查数据集。在例 9-1 中，我们先导入数据，然后选择出一些可用的字段，以此来开始实验。在保留的字段中仍然含有丰富的信息，如图 9-1 所示。

例 9-1　导入并过滤数据

```
>>> import pandas as pd

>>> model_df = pd.read_json('data/mag_papers_0/mag_subset20K.txt', lines=True)
>>> model_df.shape
(20000, 19)
>>> model_df.columns
Index(['abstract', 'authors', 'doc_type', 'doi', 'fos', 'id', 'issue',
       'keywords', 'lang', 'n_citation', 'page_end', 'page_start', 'publisher',
       'references', 'title', 'url', 'venue', 'volume', 'year'],
      dtype='object')

# 滤掉非英文文章，只关注几个变量
>>> model_df = model_df[model_df.lang == 'en']
...            .drop_duplicates(subset='title', keep='first')
...            .drop(['doc_type', 'doi', 'id', 'issue', 'lang', 'n_citation',
...                   'page_end', 'page_start', 'publisher', 'references',
...                   'url', 'venue', 'volume'],
...                  axis=1)
>>> model_df.shape
(10399, 6)
```

	abstract	authors	fos	keywords	title	year
0	A system and method for maskless direct write ...	NaN	[Electronic engineering, Computer hardware, En...	NaN	System and Method for Maskless Direct Write Li...	2015
1	NaN	[{'name': 'Ahmed M. Alluwaimi'}]	[Biology, Virology, Immunology, Microbiology]	[paratuberculosis, of, subspecies, proceedings...	The dilemma of the Mycobacterium avium subspec...	2016

图 9-1：微软学术图谱数据集的前两行

从表 9-1 中可以非常清楚地看出，需要何种程度的数据整理才能将原始数据转换为更适合建模的形式。列表和字典便于数据存储，但如果不经过一些解包操作的话，就不够整洁，不能很好地适合机器学习（Wickham, 2014）。

表9-1：model_df的数据模式

字段名称	描　述	字段类型	空值数
abstract	论文摘要	字符串	4393
authors	作者姓名与单位	字典列表，键为 name, org	1
fos	研究领域	字符串列表	1733
keywords	关键字	字符串列表	4294
title	论文题目	字符串	0
year	发表年份	整数	0

在例 9-2 中，我们先重点关注两个字段，将它们从列表和整数转换为特征数组，如图 9-2 所示。

例 9-2　协同过滤阶段 1：建立项目特征矩阵

```
>>> unique_fos = sorted(list({feature
...                           for paper_row in model_df.fos.fillna('0')
...                           for feature in paper_row }))

>>> unique_year = sorted(model_df['year'].astype('str').unique())
>>> def feature_array(x, var, unique_array):
...     row_dict = {}
...     for i in x.index:
...         var_dict = {}
...         for j in range(len(unique_array)):
...             if type(x[i]) is list:
...                 if unique_array[j] in x[i]:
...                     var_dict.update({var + '_' + unique_array[j]: 1})
...                 else:
...                     var_dict.update({var + '_' + unique_array[j]: 0})
...             else:
...                 if unique_array[j] == str(x[i]):
...                     var_dict.update({var + '_' + unique_array[j]: 1})
...                 else:
...                     var_dict.update({var + '_' + unique_array[j]: 0})
...         row_dict.update({i : var_dict})
...     feature_df = pd.DataFrame.from_dict(row_dict, dtype='str').T
...     return feature_df

>>> year_features = feature_array(model_df['year'], unique_year)
>>> fos_features = feature_array(model_df['fos'], unique_fos)

>>> first_features = fos_features.join(year_features).T

>>> from sys import getsizeof
>>> print('Size of first feature array: ', getsizeof(first_features))
Size of first feature array: 2583077234
```

	0	1	2	5	7	8	9	10	11	12	...	19985	19986	19987	19988	19993	19994	19995	19997	19998	19999
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
0-10 V lighting control	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1/N expansion	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
10G-PON	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
14-3-3 protein	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 10399 columns

图 9-2：first_features 的头部——原始数据集中观测（论文）的索引是列，特征是行

我们成功地将一个较小的数据集（大约 1 万行原始数据）转换成了 2.5GB 的特征。但对于需要快速迭代的数据探索过程来说，这种方法太笨重了。我们需要更快速的方法，使得出的特征占用更少的计算资源和实验时间。

稍安勿躁，不妨先看一下，现在这种特征在下一阶段能为我们做出多么好的推荐（见例 9-3）。我们定义“好”的推荐就是与输入相似的论文。

例 9-3　协同过滤阶段 2：查找相似项目

```
>>> from scipy.spatial.distance import cosine

>>> def item_collab_filter(features_df):
...     item_similarities = pd.DataFrame(index = features_df.columns,
...                                      columns = features_df.columns)
...     for i in features_df.columns:
...         for j in features_df.columns:
...             item_similarities.loc[i][j] = 1 - cosine(features_df[i],
...                                                       features_df[j])
...     return item_similarities

>>> first_items = item_collab_filter(first_features.loc[:, 0:1000])
```

我们只是使用两个特征来计算项目相似度，为什么计算时间如此之长？因为我们使用了嵌套 `for` 循环来计算一个 10 399 × 1000 的矩阵的点积。如果向模型中添加了更多观测，那每次循环的时间还会增加。注意，我们只筛选出了英文论文，这只是整个可用数据集的一个子集。当得到一个差不多“好”的结果时，还需要回到更大的数据集合上进行测试，看看这是不是最好的结果。

怎么才能做得更快一些呢？因为每次只需要一个结果，所以可以修改一下函数，指定我们需要的前几个结果的数量，每次只计算一个项目。我们以后会这么做，因为需要持续改进实验。眼下还是使用全特征空间，理解一下在实际数据集上使用暴力算法时迭代造成的影响。

要得到好的推荐，需要一种更好的特征转换方法。我们有足够的观测来改进吗？让我们绘制一幅热图（见例 9-4），看看是否有彼此相似的论文。结果显示在图 9-3 中。

例 9-4　论文推荐热图

```
>>> import matplotlib.pyplot as plt
>>> import seaborn as sns
>>> import numpy as np
>>> %matplotlib inline
>>> sns.set()
>>> ax = sns.heatmap(first_items.fillna(0),
...                  vmin=0, vmax=1,
...                  cmap="YlGnBu",
...                  xticklabels=250, yticklabels=250)
>>> ax.tick_params(labelsize=12)
```

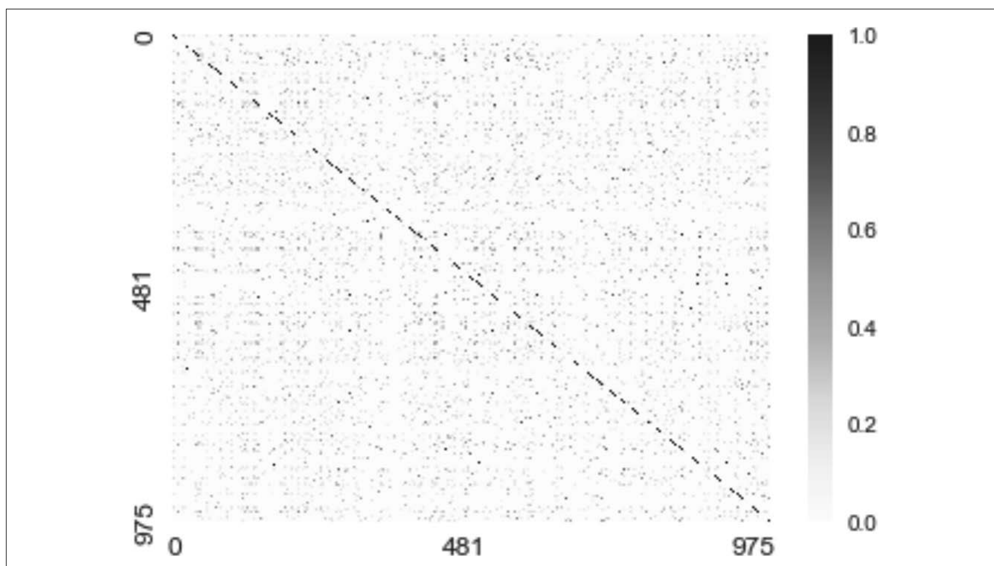


图 9-3：相似论文热图，基于两个初始特征：发表年份和研究领域

颜色较暗的像素表示彼此相似的项目。黑色对角线说明了余弦相似度能正确地表示出每篇论文都与它本身是最相似的。但是，因为有个特征中有很多 NaN 值，所以对角线是断断续续的。可以看出，尽管多数项目是彼此不相似的（这说明我们的数据集来源非常广泛），但还是有一些相似度评分很高的候选值。定性地看，这些可能是好的推荐，也可能不是，但至少说明了我们的方法不是一无是处。

例 9-5 给出了将项目相似度转换为推荐的方法。好消息是我们还有大量可用的特征，改进空间非常大。

例 9-5　基于项目的协同过滤推荐

```
>>> def paper_recommender(paper_ix, items_df):
...     print('Based on the paper: \nindex = ', paper_ix)
...     print(model_df.iloc[paper_ix])
...     top_results = items_df.loc[paper_ix].sort_values(ascending=False).head(4)
```

```
...     print('\nTop three results: ')
...     order = 1
...     for i in top_results.index.tolist()[-3:]:
...         print(order,'. Paper index = ', i)
...         print('Similarity score: ', top_results[i])
...         print(model_df.iloc[i], '\n')
...         if order < 5: order += 1

>>> paper_recommender(2, first_items)

Based on the paper:
index =  2
abstract                                                  NaN
authors     [{'name': 'Jovana P. Lekovich', 'org': 'Weill ...
fos                                                       NaN
keywords                                                  NaN
title       Should endometriosis be an indication for intr...
year                                                     2015
Name: 2, dtype: object

Top three results:
1 . Paper index =  2
Similarity score:  1.0
abstract                                                  NaN
authors     [{'name': 'Jovana P. Lekovich', 'org': 'Weill ...
fos                                                       NaN
keywords                                                  NaN
title       Should endometriosis be an indication for intr...
year                                                     2015
Name: 2, dtype: object

2 . Paper index =  292
Similarity score:  1.0
abstract                                                  NaN
authors     [{'name': 'John C. Newton'}, {'name': 'Beers M...
fos         [Wide area multilateration, Maneuvering speed,...
keywords                                                  NaN
title                    Automatic speed control for aircraft
year                                                     1955
Name: 561, dtype: object

3 . Paper index =  593
Similarity score:  1.0
abstract    This paper demonstrates that on    ite greywater...
authors     [{'name': 'Eran Friedler', 'org': 'Division of...
fos         [Public opinion, Environmental Engineering, Wa...
keywords    [economic analysis, tratamiento desperdicios, ...
title       The water saving potential and the socio-econo...
year                                                     2008
Name: 1152, dtype: object
```

呀，好消息是返回的最相似论文就是我们要找的论文，坏消息是另外两篇论文似乎与我们的搜索初衷相去甚远，即使我们使用了选定的特征。

“是的，是的，”你可能会说，“但现在是大数据时代，这会解决我们的问题！我们难道不能通过更多数据找出更好的结果吗？”可能会，但即使大数据也不能弥补糟糕的数据和特征选择所造成的恶果。

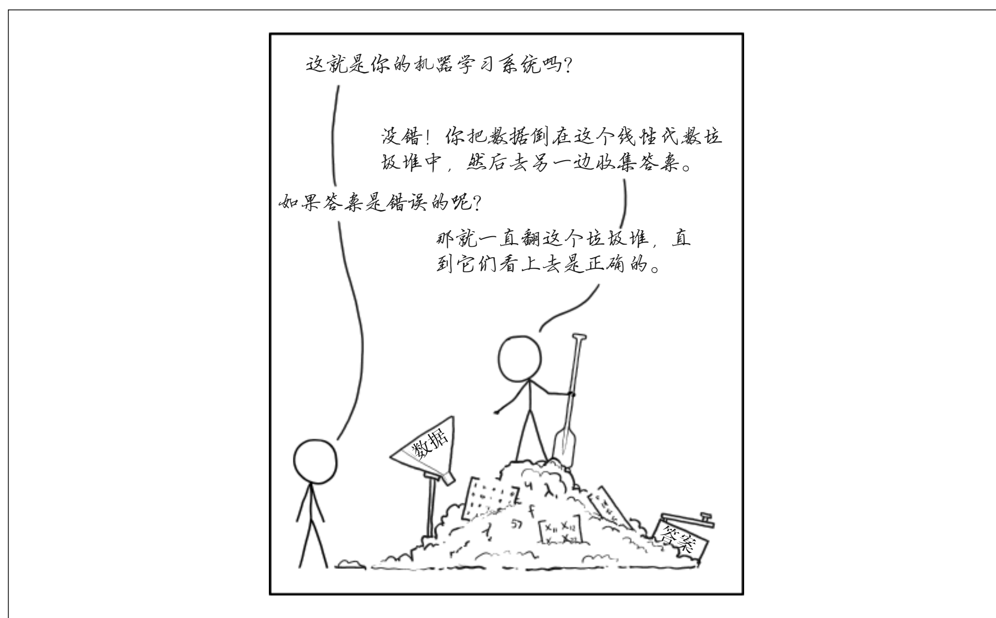


图 9-4：机器学习（https://xkcd.com/1838/）

现在的暴力方法太慢了，远算不上是智能的、迭代的特征工程。下面试验一下新的特征工程技术，看看是否能提高计算速度，找到更合适的特征和搜索结果的更好方式。

9.3 第二关：更多特征工程和更智能的模型

最初的方法是创建一个巨大的、稀疏的数组，然后通过一个筛选器暴力求解。有多种方式可以改进这种方法。下一步的重点是使用更好的技术来处理两个初始特征，并修改基于项目的协同过滤方法来加快迭代。

首先，在假设中的两个变量上，试验一下本书介绍过的精彩的特征工程技巧。在更加深入地研究了特征之后，我们可以选择那些适合每种变量的技术，将变量转换为适合推荐系统的“更好”的特征。

学术论文推荐器：第2轮

先看**出版年份**。2.2.2 节中介绍了为什么使用原始计数作为特征不适合那些使用相似度度量的方法。例 9-6 和图 9-5 会研究如何对 `year` 进行转换，以使它更加适合我们选择的模型。

例 9-6　等宽分箱 + 虚拟编码（第 1 部分）

```
>>> print("Year spread: ", model_df['year'].min()," - ", model_df['year'].max())
>>> print("Quantile spread:\n", model_df['year'].quantile([0.25, 0.5, 0.75]))
Year spread:  1831  -  2017
Quantile spread:
0.25    1990.0
0.50    2005.0
0.75    2012.0
Name: year, dtype: float64

# 绘制出版年份，看一下它的分布
>>> fig, ax = plt.subplots()
>>> model_df['year'].hist(ax=ax,
...                       bins= model_df['year'].max() - model_df['year'].min())
>>> ax.tick_params(labelsize=12)
>>> ax.set_xlabel('Year Count', fontsize=12)
>>> ax.set_ylabel('Occurrence', fontsize=12)
```

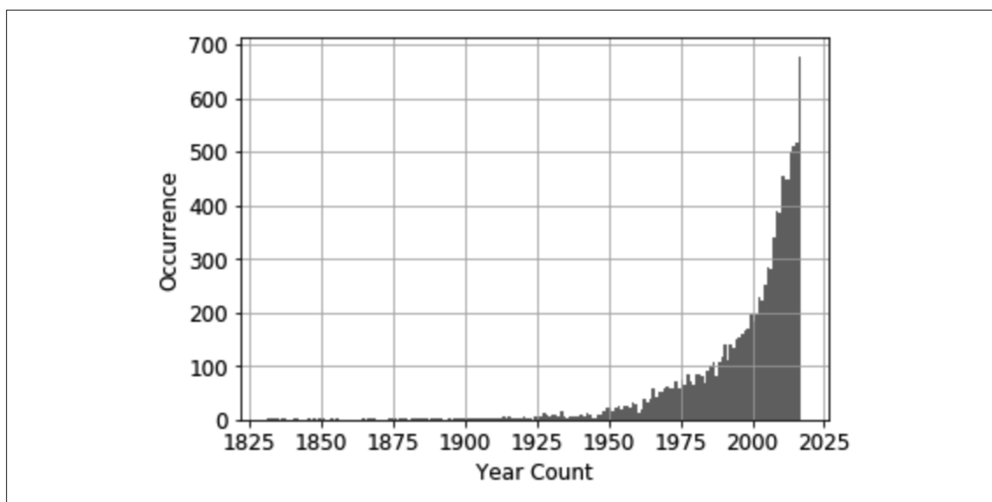


图 9-5：数据集中 10 000 多篇学术论文的原始出版年份分布

从图 9-5 中的偏态分布来看，出版年份非常适合分箱操作。

我们将根据变量的取值范围来分箱，而不是唯一的特征值数量。为了进一步压缩特征空间，我们对分箱结果进行虚拟编码（见例 9-7）。Pandas 的内置函数可以完成这两项任务。这些方法的结果很容易解释，所以我们可以对转换后的特征做一个快速检查（见图 9-6），再进行后面的工作。

例 9-7　定宽分箱 + 虚拟编码（第 2 部分）

```
# 按照10年的宽度进行分箱，可将出版年份特征空间从156压缩到19
>>> bins = int(round((model_df['year'].max() - model_df['year'].min()) / 10))

>>> temp_df = pd.DataFrame(index = model_df.index)
```

```
>>> temp_df['yearBinned'] = pd.cut(model_df['year'].tolist(), bins, precision = 0)
>>> X_yrs = pd.get_dummies(temp_df['yearBinned'])
>>> X_yrs.columns.categories
IntervalIndex([(1831.0, 1841.0], (1841.0, 1851.0], (1851.0, 1860.0],
               (1860.0, 1870.0], (1870.0, 1880.0] ... (1968.0, 1978.0],
               (1978.0, 1988.0], (1988.0, 1997.0], (1997.0, 2007.0],
               (2007.0, 2017.0]]
              closed='right',
              dtype='interval[float64]')

# 绘制新的分布
>>> fig, ax = plt.subplots()
>>> X_yrs.sum().plot.bar(ax = ax)
>>> ax.tick_params(labelsize=8)
>>> ax.set_xlabel('Binned Years', fontsize=12)
>>> ax.set_ylabel('Counts', fontsize=12)
```

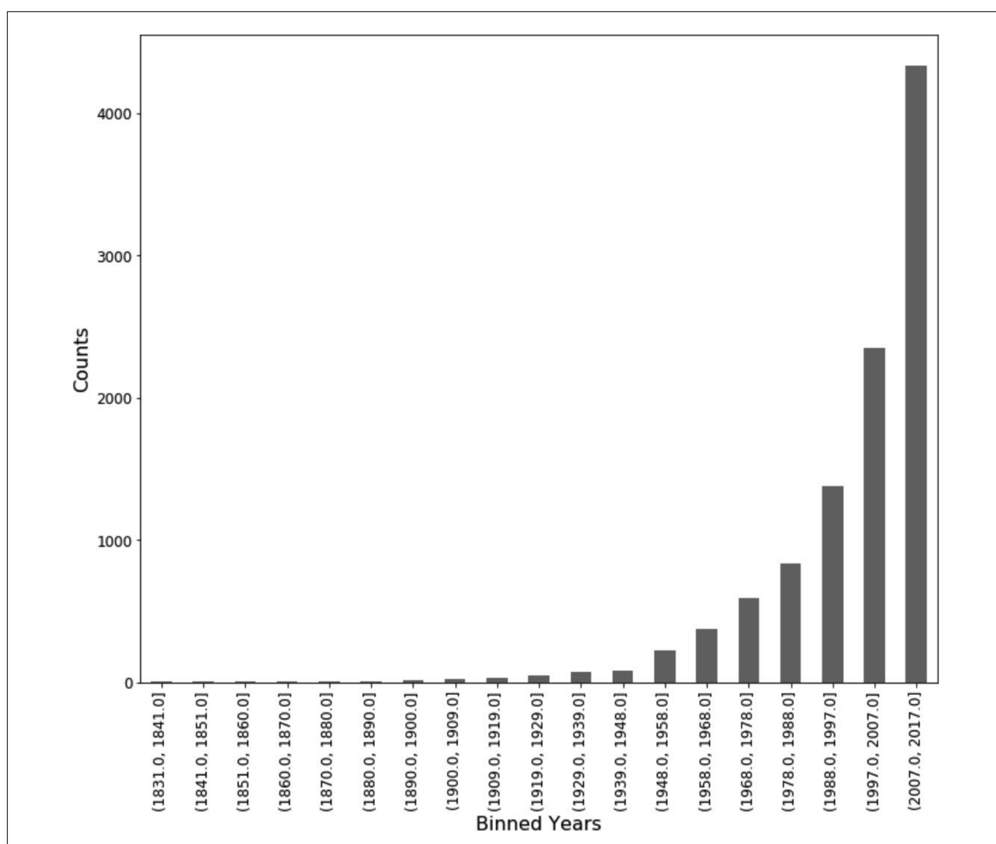


图 9-6：分箱之后新特征 X_yrs 的分布

按照 10 年的宽度进行分箱，我们保留了原始变量中的基本分布。如果需要对其他分布进行分箱，可以修改一下分箱设置，改变变量在模型中的呈现方式。因为我们使用的是余弦

相似度，所以这样做没有问题。下面接着处理模型中最初包含的另外一个特征。

研究领域特征空间对初始模型的大小和处理时间有非常显著的影响。

检查一下已经完成的工作。通过解析字符串列表，我们在第一关中创建了一个“短语袋”。既然已经有了一个非常好用的稀疏数组，就应该使用这个更高效的数据类型来表示这个短语袋。例 9-8 演示了将 Pandas 数据框转换为 NumPy 稀疏数组之后对计算时间的影响。

例 9-8　将短语袋从 pd.Series 转换为 NumPy 稀疏数组

```
>>> X_fos = fos_features.values

# 看一下每个对象的大小，就能知道这种转换对以后的操作有什么影响。
>>> print('Our pandas Series, in bytes: ', getsizeof(fos_features))
>>> print('Our hashed numpy array, in bytes: ', getsizeof(X_fos))

Our pandas Series, in bytes:  2530632380
Our hashed numpy array, in bytes:  112
```

真是太棒了！把两个特征组合在一起，一起输入过滤器（见例 9-9），重新运行推荐器（见例 9-10）看看能否得到更好的结果。在过滤器中，我们使用了 scikit-learn 的余弦相似度函数。我们还是每次只对一个项目进行推荐，目的是节省计算时间。

例 9-9　协同过滤阶段 1+2：建立项目特征矩阵，搜索相似项目

```
>>> second_features = np.append(X_fos, X_yrs, axis = 1)
>>> print("The power of feature engineering saves us, in bytes: ",
...       getsizeof(first_features) - getsizeof(second_features))
The power of feature engineering saves us, in bytes:  168066769

>>> from sklearn.metrics.pairwise import cosine_similarity

>>> def piped_collab_filter(features_matrix, index, top_n):
...     item_similarities = \
...         1 - cosine_similarity(features_matrix[index:index+1],
...                               features_matrix).flatten()
...     related_indices = \
...         [i for i in item_similarities.argsort()[::-1] if i != index]
...     return [(index, item_similarities[index])
...             for index in related_indices
...            ][0:top_n]
```

例 9-10　基于项目的协同过滤推荐：第 2 轮

```
>>> def paper_recommender(items_df, paper_ix, top_n):
...     if paper_ix in model_df.index:
...         print('Based on the paper:')
...         print('Paper index = ', model_df.loc[paper_ix].name)
...         print('Title :', model_df.loc[paper_ix]['title'])
...         print('FOS :', model_df.loc[paper_ix]['fos'])
...         print('Year :', model_df.loc[paper_ix]['year'])
...         print('Abstract :', model_df.loc[paper_ix]['abstract'])
...         print('Authors :', model_df.loc[paper_ix]['authors'], '\n')
```

```
...         # 通过数据框索引定义需要的数组位置索引
...         array_ix = model_df.index.get_loc(paper_ix)
...         top_results = piped_collab_filter(items_df, array_ix, top_n)
...         print('\nTop',top_n,'results: ')

...         order = 1
...         for i in range(len(top_results)):
...             print(order,'. Paper index = ',
...                   model_df.iloc[top_results[i][0]].name)
...             print('Similarity score: ', top_results[i][1])
...             print('Title :', model_df.iloc[top_results[i][0]]['title'])
...             print('FOS :', model_df.iloc[top_results[i][0]]['fos'])
...             print('Year :', model_df.iloc[top_results[i][0]]['year'])
...             print('Abstract :', model_df.iloc[top_results[i][0]]['abstract'])
...             print('Authors :', model_df.iloc[top_results[i][0]]['authors'],
...                   '\n')
...             if order < top_n: order += 1
...     else:
...         print('Whoops! Choose another paper. Try something from here: \n',
...               model_df.index[100:200])

>>> paper_recommender(second_features, 2, 3)
Based on the paper:
Paper index =  2
Title : Should endometriosis be an indication for intracytoplasmic sperm inject ...
FOS : nan
Year : 2015
Abstract : nan
Authors : [{'name': 'Jovana P. Lekovich', 'org': 'Weill Cornell Medical College, ...

Top 3 results:
1 . Paper index =  10055
Similarity score:  1.0
Title : [Diagnosis of cerebral tumors; comparative studies on arteriography, ...
FOS : ['Radiology', 'Pathology', 'Surgery']
Year : 1953
Abstract : nan
Authors : [{'name': 'Antoine'}, {'name': 'Lepoire'}, {'name': 'Schoumacker'}]

2 . Paper index =  11771
Similarity score:  1.0
Title : A Study of Special Functions in the Theory of Eclipsing Binary Systems
FOS : ['Contact binary']
Year : 1981
Abstract : nan
Authors : [{'name': 'Filaretti Zafiropoulos', 'org': 'University of Manchester'}]

3 . Paper index =  11773
Similarity score:  1.0
Title : Studies of powder flow using a recording powder flowmeter and measure ...
FOS : nan
Year : 1985
Abstract : This paper describes the utility of the dynamic measurement of the ...
Authors : [{'name': 'Ramachandra P. Hegde', 'org': 'Department of Pharmacy, ...
```

说实话，我并不认为这次特征选择的效果有多么好。这些字段中有很多缺失值。下面继续看一下能否找出一些信息更丰富的特征。

找到你的位置

在 Pandas 数据框和 NumPy 矩阵之间的转换中，索引会令人迷惑——索引的大小相同，但其分配的位置却不一样。为了解决这个问题，Pandas 提供了 `.iloc`、`.loc` 和 `.get_loc` 三种方法，如例 9-11 所示。

- `.loc` 返回基于初始 Pandas 数据框的索引，可以让我们引用具体的论文。
- `.iloc` 使用整数位置，和 NumPy 数组的索引是一样的。
- `.get_loc` 可以帮助我们在已知数据框索引时找出整数位置。

例 9-11　在转换时维护索引分配

```
>>> model_df.loc[21]
abstract    A microprocessor includes hardware registers t...
authors                      [{'name': 'Mark John Ebersole'}]
fos         [Embedded system, Parallel computing, Computer...
keywords                                                  NaN
title       Microprocessor that enables ARM ISA program to...
year                                                     2013
Name: 21, dtype: object

>>> model_df.iloc[21]
abstract                                                  NaN
authors     [{'name': 'Nicola M. Heller'}, {'name': 'Steph...
fos         [Biology, Medicine, Post-transcriptional regul...
keywords    [glucocorticoids, post transcriptional regulat...
title       Post-transcriptional regulation of eotaxin by ...
year                                                     2002
Name: 30, dtype: object

>>> model_df.index.get_loc(30)
21
```

9.4　第三关：更多特征=更多信息

到此为止，我们的实验并没有支持初始假设，即仅靠**出版年份**和**研究领域**就足以推荐出相似的论文。既然如此，有以下几种选择。

- 使用原始数据集中更多的数据，看看能否得到更好的结果。
- 花费更多时间去探索数据，看看能否找到一个足够密集的集合来提供好的推荐。
- 添加更多特征，继续迭代当前模型。

第一种选择假设问题在于我们对数据的抽样，这是有可能的，但这样做和图 9-4 中翻动数据垃圾堆找寻更好答案的比喻是一样的。

第二种选择可以更好地理解原始数据。这应该在数据探索过程中，根据特征和模型选择决策的变化不断地重新进行。在本例中，初始的子样本选择就反映了这个步骤。因为在数据集中还有更多变量可用，所以我们就不再重新进行这个步骤了。

最后看第三种选择，添加更多特征，在当前模型的基础上继续前进。加入更多关于每个项目的信息可以改善相似度评分，进而得到更好的推荐。

根据我们最初的探索，下一步的工作将集中在信息量最丰富的两个字段上：**论文摘要**和**作者姓名**。

学术论文推荐器：第3轮

回顾一下第 4 章，我们可以知道论文摘要非常适合使用 tf-idf 来过滤掉噪声并找出显著相关的单词。我们在例 9-12 中实现了 tf-idf。

例 9-12　停用词 +tf-idf

```
# 需要滤掉NaN值，供sklearn以后使用
>>> filled_df = model_df.fillna('None')

>>> from sklearn.feature_extraction.text import TfidfVectorizer

>>> vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5,
...                              stop_words='english')
>>> X_abstract = vectorizer.fit_transform(filled_df['abstract'])
>>> third_features = np.append(second_features, X_abstract.toarray(), axis = 1)
```

论文作者比较混乱，而且参差不齐，我们把它整理成字典，再对它进行 one-hot 编码，这样可以降低计算负载，如例 9-13 所示。

例 9-13　使用 scikit-learn 的 `DictVectorizer` 进行 one-hot 编码

```
>>> authors_list = []

>>> for row in filled_df.authors.itertuples():
...     # 为每个序列索引创建一个字典
...     if type(row.authors) is str:
...         y = {'None': row.Index}
...     if type(row.authors) is list:
...         # 将这些键+值添加到字典中
...         y = dict.fromkeys(row.authors[0].values(), row.Index)
...     authors_list.append(y)

>>> authors_list[0:5]
[{'None': 0},
{'Ahmed M. Alluwaimi': 1},
{'Jovana P. Lekovich': 2, 'Weill Cornell Medical College, New York, NY': 2},
{'George C. Sponsler': 5},
{'M. T. Richards': 7}]
```

```
>>> from sklearn.feature_extraction import DictVectorizer
>>> v = DictVectorizer(sparse=False)
>>> D = authors_list
>>> X_authors = v.fit_transform(D)
>>> fourth_features = np.append(third_features, X_authors, axis = 1)
```

现在可以使用推荐器看看这些新特征的效果了。例 9-14 给出了结果。

例 9-14　基于项目的协同过滤推荐：第 3 轮

```
>>> paper_recommender(fourth_features, 2, 3)

Based on the paper:
Paper index =  2
Title : Should endometriosis be an indication for intracytoplasmic sperm inject ...
FOS : nan
Year : 2015
Abstract : nan
Authors : [{'name': 'Jovana P. Lekovich', 'org': 'Weill Cornell Medical College, ...


Top 3 results:
1 . Paper index =  10055
Similarity score:  1.0
Title : [Diagnosis of cerebral tumors; comparative studies on arteriography, ...
FOS : ['Radiology', 'Pathology', 'Surgery']
Year : 1953
Abstract : nan
Authors : [{'name': 'Antoine'}, {'name': 'Lepoire'}, {'name': 'Schoumacker'}]

2 . Paper index =  5601
Similarity score:  1.0
Title : 633 Survival after coronary revascularization, with and without mitral ...
FOS : ['Cardiology']
Year : 2005
Abstract : nan
Authors : [{'name': 'J.B. Le Polain De Waroux'}, {'name': 'Anne-Catherine ...

3 . Paper index =  12256
Similarity score:  1.0
Title : Nucleotide Sequence and Analysis of an Insertion Sequence from Bacillus ...
FOS : ['Biology', 'Molecular biology', 'Insertion sequence', 'Nucleic acid ...
Year : 1994
Abstract : A 5.8-kb DNA fragment encoding the  cryIC  gene from  Bacillus thur...
Authors : [{'name': 'Geoffrey P. Smith'}, {'name': 'David J. Ellar'}, {'name': ...
```

尽管某些字段中有缺失数据，但最后一轮特征工程得到的前 3 个结果都向我们推荐了医学领域的其他论文。

这个数据集中的论文范围非常广泛。例如，论文的随机抽样中可以包括以下研究领域："Coupling constant""Evapotranspiratioin""Hash function""IVMS""Meditation""Pareto analysis""Second-generation wavelet transform""Slip"和"Spiral galaxy"。考虑到这 1 万

多篇论文中共有 7604 个唯一的研究领域，这些最终结果应该是向着正确方向前进。我们的工作正逐步接近有用的模型，我们对此非常有信心。

对更多文本型变量继续迭代，比如找出论文题目中的名词短语，或对关键字进行词干提取，都可以使我们更加接近“最佳”推荐。

需要注意的是，这里所说的“最佳”只是所有推荐器和搜索引擎追求的一种理想状态。我们要搜索出一个对用户最有帮助的结果，这不一定能从数据中直接表现出来。特征工程可以抽象出显著的特征并将其转化为一种表示形式，以使算法能揭示出其中包含的显式和隐式信息。

9.5 小结

正如你看到的，建立一个机器学习模型非常容易，但要建立一个**好**模型并得到**有用**的结果则需要花很多时间做很多工作。在本章中，我们为了获得更好的结果，检验了可能的变量集合，使用多种特征工程方法进行了试验。在这里，“更好”的含义不仅包括从训练和测试中得到好的结果，还包括使模型更简洁，以及减少在各种试验上的迭代时间。

本书开头说过，要精通一门学科，需要深入理解其中的原理，以便获得直觉，进而有效地将知识应用到工作中。希望从本书中你能获得必要的方法和工具，提高工作的效率和效果，同时扩展你的数学与计算机能力，更好地理解为什么特征工程是开发有用的机器学习模型的一项基本技能。

9.6 参考文献

Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. Item-Based Collaborative Filtering Recommendation Algorithms [C]. Proceedings of the 10th International Conference on the World Wide Web (2001) 285–295.

Sinha, Arnab, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June (Paul) Hsu, and Kuansan Wang. An Overview of Microsoft Academic Service (MAS) and Applications [C]. Proceedings of the 24th International Conference on the World Wide Web (2015): 243–246.

Tang, Jie, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. ArnetMiner: Extraction and Mining of Academic Social Networks [C]. Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2008): 990–998.

Wickham, Hadley. Tidy Data [J]. The Journal of Statistical Software 59 (2014).

附录A 线性建模与线性代数基础

A.1 线性分类概述

当有个标记好的数据集时，特征空间中散布着不同类别的数据点，分类器的工作就是将不同类别的点分隔开。分类器的实现方式是对不同类别的数据点生成不同的输出。例如，当只有两个类别时，一个好的分类器应该对一个类别生成较大的输出，而对另一个类别生成较小的输出。那些正好位于两个类别之间的点可以形成一个**决策面**（见图 A-1）。

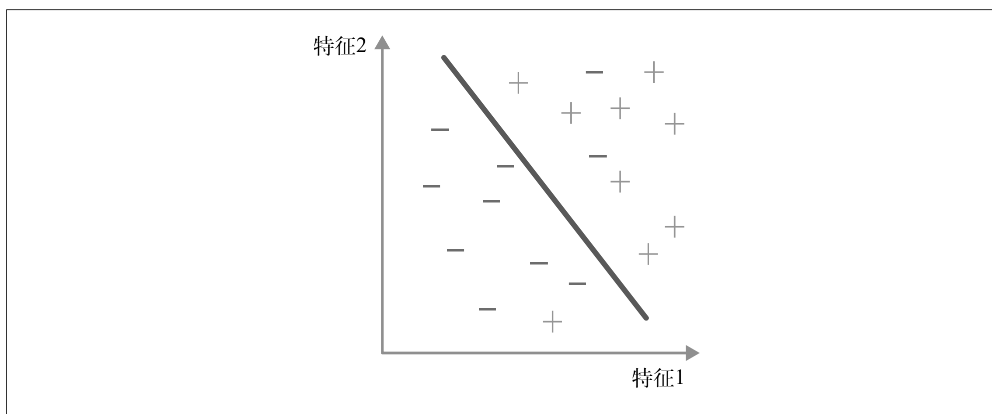


图 A-1：简单二值分类找出一个界面来分隔两个类别的数据点

可以使用很多函数作为分类器。由于若干原因，应该使用能清晰划分类别的最简单的函数。首先，相比最好的复杂分隔符，找到最好的简单分隔符更容易。其次，简单函数通常

在新数据上的扩展效果更好，因为要使新数据适合过于复杂的训练数据是非常困难的（这就是**过拟合**的概念）。简单模型或许会犯错误，比如在图 A-1 中，有些点在错误的一侧。但是，我们宁愿牺牲一些训练准确度，来换取更加简单的、能得到更高测试准确度的决策面。这种使复杂度最小而可用性最大的原则称为“奥卡姆剃刀”，在科学界和工程界都广泛适用。

最简单的函数是一条直线。带有一个输入变量的**线性函数**是最常见的（见图 A-2）。

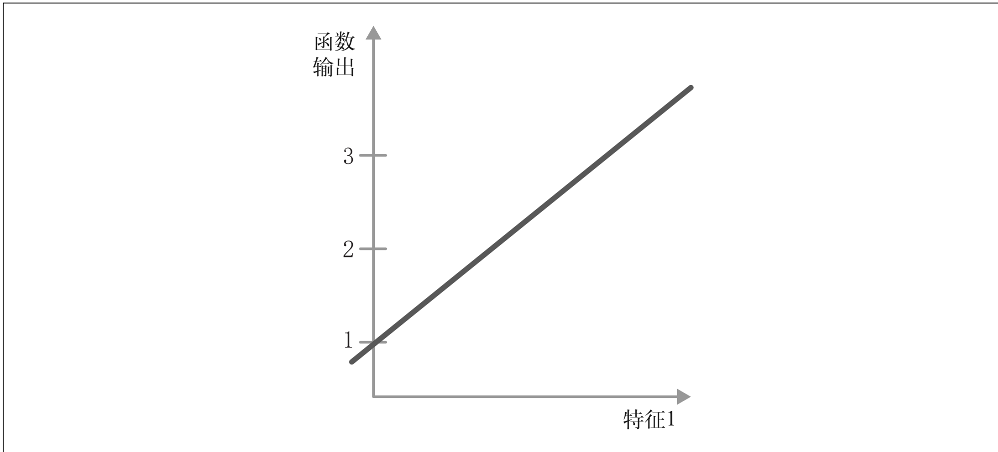


图 A-2：一个输入变量的线性函数

对于有两个输入变量的线性函数的可视化，可以使用三维空间中的平面，也可以使用二维空间中的等高线（如图 A-3 所示）。和拓扑地形图相似，在等高线图中，每条线都表示具有同样输出的输入空间中的点。

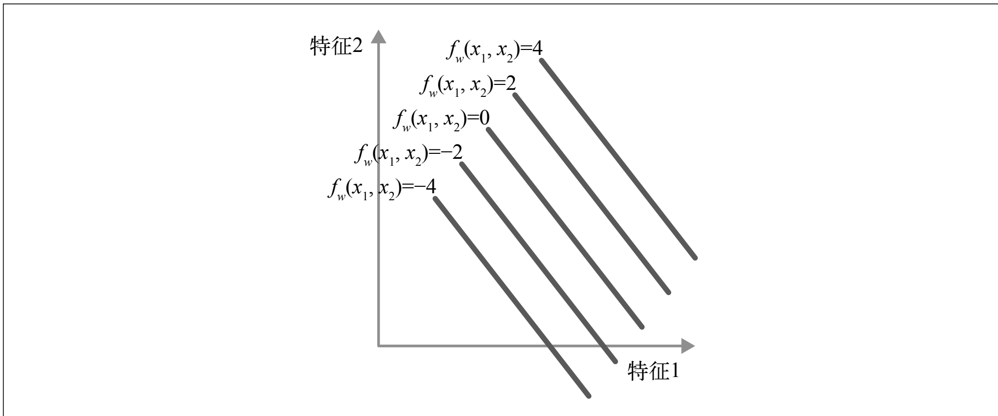


图 A-3：二维空间中线性函数的等高线图

更高维度的线性函数很难可视化。高维线性函数被称为**超平面**。但超平面很容易用代数公式表示。一个输入集合为 $x_1, x_2, \cdots, x_n$，权重参数集合为 $w_0, w_1, \cdots, w_n$ 的多维线性函数可以表示如下：

$$f_w(x_1, x_2, \cdots, x_n) = w_0 + w_1 * x_1 + w_2 * \mathrm{x}_2 + \cdots + w_n * x_n$$

使用向量表示时，公式可以更简洁：

$$f_w(\boldsymbol{x}) = \boldsymbol{x}^{\mathrm{T}} \boldsymbol{w}$$

我们遵循数学表示的常用惯例，用粗体字母表示向量，非粗体字母表示标量。向量 $\boldsymbol{x}$ 在最前面加上了一个 1，作为截距项 w_0 的占位符。如果所有输入特征都是 0，那么函数的输出就是 w_0。所以，w_0 也称为**偏差**或**截距项**。

训练一个线性分类器等价于找出类别之间的最佳分隔超平面，这个问题可以转化为找出空间中方向完全正确的最佳向量 $\boldsymbol{w}$。因为每个数据点都有一个目标标签 y，所以我们就是要找到一个 $\boldsymbol{w}$ 来直接预测这个目标标签：[1]

$$\boldsymbol{x}^{\mathrm{T}} \boldsymbol{w} = \boldsymbol{y}$$

因为通常有多个数据点，所以要找到一个 $\boldsymbol{w}$，能同时使得所有预测都接近目标标签：

$$\boldsymbol{A}\boldsymbol{w} = \boldsymbol{y}$$

其中，$\boldsymbol{A}$ 被称为**数据矩阵**（在统计学中也称为设计矩阵）。它包含特定形式的数据：每行是一个数据点，每列是一个特征。（有时人们使用它的转置形式，其中行是特征，列是数据点。）

A.2 矩阵的解析

为了求解前面的方程，需要一些基本的线性代数知识。如果想系统地学习线性代数，我们强烈推荐 Strang (2006)。

这个方程阐述了这一事实：一个特定的矩阵乘以一个特定的向量，可以得到一个具体的结果。矩阵也称为线性算子，这个名称更加清楚地说明了矩阵就是一台小型机器。这台机器接受一个向量作为输入，然后使用若干种核心操作的组合生产出另一个向量。这些核心操作包括：旋转向量的方向，增加或减少向量的维度，拉伸或压缩向量的长度。在控制输入空间的形状方面，这些操作的功能非常强大。

注 1：严格地说，这里给出的公式是用来做线性回归的，不是线性分类。区别在于，回归问题可以使用实数型的目标变量，而分类问题的目标变量通常都是表示不同类别的整数。通过非线性变换，回归器可以转换为分类器。例如，逻辑回归分类器将输入的线性变换传递到一个逻辑函数中。这种模型称为**广义线性模型**，它们的内核是线性函数。尽管这个例子是关于分类的，我们还是使用线性回归的公式作为一种教学工具，因为它特别容易分析。它的结论可以很容易地推广到广义线性分类器中。

举个例子，如图 A-4 所示，一个 3×2 的矩阵可以将一个二维正方形转换为三维空间中的菱形。它将输入空间中的每个向量进行了旋转和拉伸，在输出空间形成了新的向量。

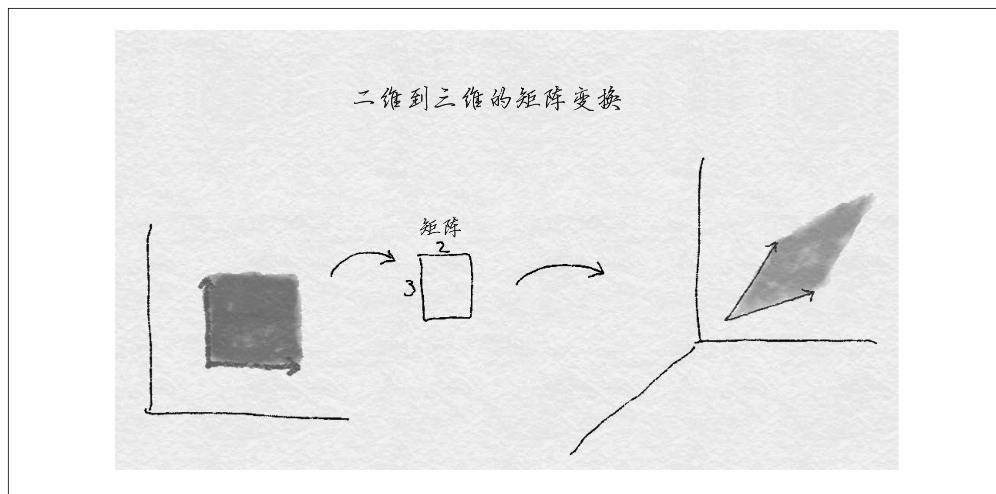


图 A-4：二维到三维的矩阵变换

A.2.1　从向量到子空间

为了理解线性算子，必须看看它是如何将输入转换为输出的。幸运的是，我们不用每次只分析一个输入向量。向量可以构成子空间，**线性算子可以操作向量子空间**。

子空间是满足两个条件的一组向量。首先，如果它包含一个向量，那么就包含从原点到这个点的直线。其次，如果它包含两个点，那么就包含这两个向量的所有线性组合。线性组合是两种类型操作的组合：用标量乘以向量，以及两个向量相加。

子空间的一个重要特性是**秩**，或称维度，它是这个空间中自由度的度量。直线的秩是 1，二维平面的秩是 2，以此类推。如果你能想象出多维空间中一只多维度的鸟，那么子空间的秩就可以告诉我们这只鸟可以沿着多少个“独立的”方向自由飞翔。这里的“独立”指的是“线性独立”：如果一个向量不能表示成另一个向量与一个常数的乘积形式，就可以说这两个向量是线性独立的（也就是说，这两个向量的方向不是完全相同，也不是完全相反）。

子空间可以定义为一组**基向量**的张成。（张成是个技术术语，用来描述由一组向量的所有线性组合构成的集合。）一组向量的张成就是它的线性组合（因为就是这么定义的）。所以，如果我们有一组基向量，就可以用任意非零常数乘以这些向量，或把它们相加，得到另一组基。

如果有更加唯一和可识别的基来描述子空间，那将是非常好的。由一组单位长度而且彼此正交的向量组成的基称为**正交基**。正交也是一个技术术语。（在所有数学和科学书籍中，

至少有 50% 的词语是技术术语。如果你不相信，可以对本书做一次词袋计数。）如果两个向量的内积为 0，它们就是彼此**正交**的。从各种意义上说，我们都可以认为正交向量是彼此成 90 度角的。（这在欧几里得空间中是成立的，欧几里得空间非常类似于我们实际生活的三维空间。）将正交向量归一化，使它具有单位长度，就可以将它们转换为一组规则的标尺。

总的来说，子空间就像是一顶帐篷，正交基向量就是那些能够撑起帐篷的、彼此成直角的柱子。正交基向量的总数就是子空间的秩。图 A-5 演示了这些概念。

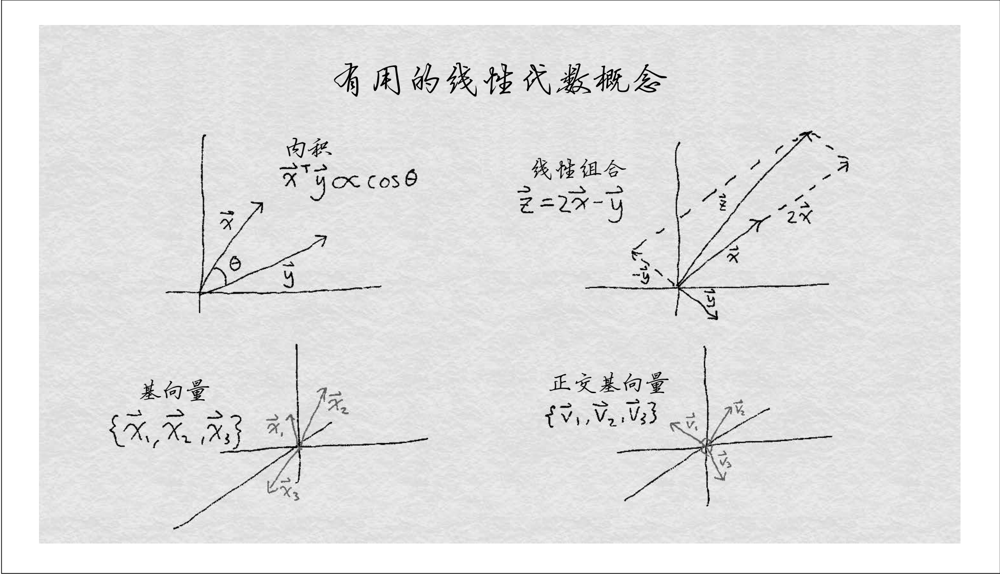


图 A-5：四个有用的线性代数概念演示：内积、线性组合、基向量和正交基向量

有用的线性代数定义

对于那些喜欢用数学来进行思考的人，下面是一些能使我们的描述更精确的数学定义。

标量

一个数值 c，与向量相对。

向量

$$\boldsymbol{x} = (x_1, x_2, \cdots, x_n)$$

线性组合

$$a\boldsymbol{x} + b\boldsymbol{y} = (ax_1 + by_1, ax_2 + by_2, \cdots, ax_n + by_n)$$

一组向量 $v_1, \cdots, v_k$ 的张成

对于任意 $a_1, \cdots, a_k$，向量 $\boldsymbol{u} = a_1\boldsymbol{v}_1 + \cdots + a_k\boldsymbol{v}_k$ 的集合。

线性独立

$\boldsymbol{x}$ 和 $\boldsymbol{y}$ 是线性独立的，如果对任意标量常数 c 都有 $\boldsymbol{x} \neq c\boldsymbol{y}$。

内积

$$<\boldsymbol{x},\boldsymbol{y}> = x_1y_1 + x_2y_2 + \cdots + x_ny_n$$

正交向量

两个向量 $\boldsymbol{x}$ 和 $\boldsymbol{y}$ 是正交的，如果 $<\boldsymbol{x},\boldsymbol{y}> = 0$。

子空间

一个更大向量空间中的向量子集，满足以下 3 个条件。

(1) 包含 0 向量。
(2) 如果包含向量 $\boldsymbol{v}$，则包含所有向量 $c\boldsymbol{v}$，其中 c 是个标量。
(3) 如果包含两个向量 $\boldsymbol{u}$ 和 $\boldsymbol{v}$，则包含向量 $\boldsymbol{u} + \boldsymbol{v}$。

基

能张成一个子空间的向量集合。

正交基

基 $\{\boldsymbol{v}_1,\boldsymbol{v}_2,\cdots,\boldsymbol{v}_n\}$，其中对于所有 i,j，都有 $<\boldsymbol{v}_i, \boldsymbol{v}_j> = 0$。

子空间的秩

能张成子空间的线性独立基向量的最小数目。

A.2.2　奇异值分解（SVD）

矩阵对输入向量进行线性变换。线性变换非常简单和受限，由此可知矩阵不能随意地操作子空间。线性代数中一个最有趣的定理说明，所有方阵（不管其中包含什么数值）都可以将一组特定向量通过某种放缩映射回它们本身。对更为一般的长方形矩阵，它可以将一组输入向量映射为相应的输出向量，它的**转置**可以将这些输出向量映射回原来的输入向量。用数学术语表示就是方阵具有与特征值对应的特征向量，长方形矩阵具有与奇异值对应的左奇异向量和右奇异向量。

特征向量与奇异向量

令 $\boldsymbol{A}$ 是一个 $n \times n$ 矩阵。如果存在一个向量 $\boldsymbol{v}$ 和一个标量 λ，使得 $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$，则称 $\boldsymbol{v}$ 是 $\boldsymbol{A}$ 的一个**特征向量**，λ 是 $\boldsymbol{A}$ 的一个**特征值**。

令 $\boldsymbol{A}$ 是一个长方形矩阵。如果存在向量 $\boldsymbol{u}$ 和向量 $\boldsymbol{v}$ 以及一个标量 σ，使得 $\boldsymbol{A}\boldsymbol{v} = \sigma\boldsymbol{u}$ 且 $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{u} = \sigma\boldsymbol{v}$，则称 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 是 $\boldsymbol{A}$ 的**左奇异向量**和**右奇异向量**，σ 是 $\boldsymbol{A}$ 的一个**奇异值**。

矩阵奇异值分解的数学表示形式如下：

$$A = U\Sigma V^{\mathrm{T}}$$

其中矩阵 U 和 V 的列分别构成了输入空间和输出空间的正交基，Σ 是一个包含奇异值的对角阵。

从几何上看，矩阵依次进行了以下变换。

(1) 将输入向量映射到右奇异基向量。
(2) 通过相应的奇异值对每个坐标进行放缩。
(3) 用放缩结果乘以每个左奇异向量。
(4) 累加结果。

图 A-6 给出了示意图。操作从右向左进行矩阵和向量的相乘。最右侧的矩阵对输入进行旋转，并隐式地投影到一个低维空间。在图 A-6 中，输入立方体变成了一个平面正方形，而且是旋转过的。下一个矩阵在一个方向上挤压正方形，并在另一个方向上拉伸它，使正方形变成了长方形。最后，最左侧的矩阵再次旋转长方形，并对其投影，回到一个可能更高维的空间——但它还是一个平面长方形，不是某种高维对象。

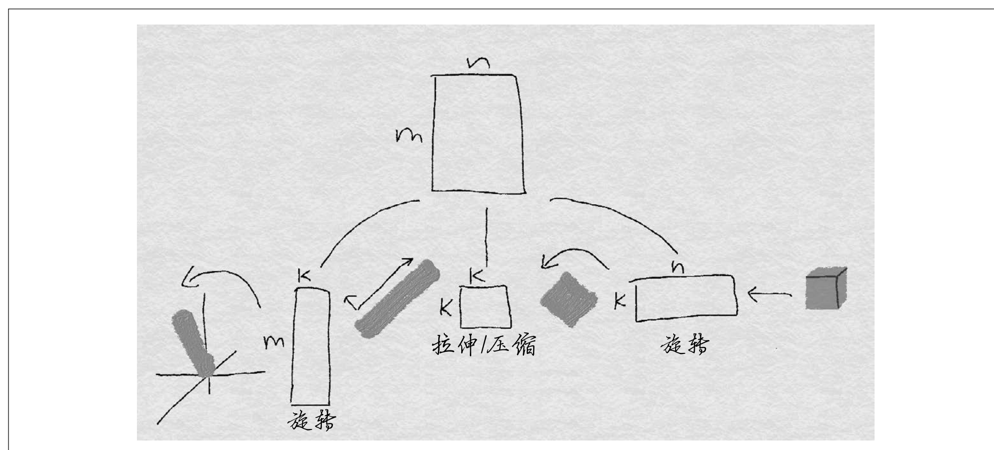


图 A-6：矩阵分解为三个小矩阵：旋转、放缩、旋转

如果 A 是个实矩阵（也就是说所有矩阵元素都是实数），那么所有奇异值和奇异向量也都是实数。奇异值可以是正数、负数或 0。矩阵奇异值的有序集合称为它的**谱**，它可以揭示大量矩阵信息。奇异值之间的差影响着解的稳定性，最大和最小的奇异值绝对值之间的比率（**条件数**）影响着一个迭代求解器找到矩阵解的速度。这些特性对我们能找到的解的质量都有非常大的影响。

A.2.3 数据矩阵的四个基本子空间

通过四个基本子空间解析矩阵是另一种有用的矩阵解析方式，这四个基本子空间是：列空

间、行空间、零空间和左零空间。这四个子空间可以完整地描述由 $\boldsymbol{A}$ 和 $\boldsymbol{A}^{\mathrm{T}}$ 定义的线性系统的解（并由此得名）。

对于数据矩阵（其中行是数据点，列是特征），这四个基本子空间可以通过数据与特征之间的关联来理解。下面详细地进行说明。

1. 列空间

数学定义：

输出向量 $\boldsymbol{s}$ 的集合，其中 $\boldsymbol{s} = \boldsymbol{A}\boldsymbol{w}$，权重向量 $\boldsymbol{w}$ 是可变的。

数学解释：

列的所有可能的线性组合。

数据解释：

基于观测特征，所有结果都是线性可预测的。向量 $\boldsymbol{w}$ 中包含了每个特征的权重。

基：

对应于非零奇异值的左奇异向量（矩阵 $\boldsymbol{U}$ 的列子集）。

2. 行空间

数学定义：

输出向量 $\boldsymbol{r}$ 的集合，其中 $\boldsymbol{r} = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{A}$，权重向量 $\boldsymbol{u}$ 是可变的。

数学解释：

行的所有可能的线性组合。

数据解释：

行空间中的一个向量可以表示为现有数据点的一个线性组合。因此，行空间可以解释为“非奇异”数据的空间。向量 $\boldsymbol{u}$ 中包含了线性组合中每个数据点的权重。

基：

对应于非零奇异值的右奇异向量（矩阵 $\boldsymbol{V}$ 的列子集）。

3. 零空间

数学定义：

输入向量 $\boldsymbol{w}$ 的集合，其中 $\boldsymbol{w}$ 满足 $\boldsymbol{A}\boldsymbol{w} = 0$。

数学解释：

与 $\boldsymbol{A}$ 中所有行都正交的向量集合。零空间可以被矩阵挤压为 0，它是添加在 $\boldsymbol{A}\boldsymbol{w} = \boldsymbol{y}$ 的解空间上的“泡沫”。

数据解释：

不能表示为现有数据点的线性组合的“奇异”数据点。

基：

对应于 0 奇异值的右奇异向量（$\boldsymbol{V}$ 中的其余列）。

4. 左零空间

数学定义：

输入向量 $\boldsymbol{u}$ 的集合，其中 $\boldsymbol{u}$ 满足 $\boldsymbol{u}^{\mathrm{T}}\boldsymbol{A} = 0$。

数学解释：

与 $\boldsymbol{A}$ 中所有列都正交的向量集合。左零空间与列空间是正交的。

数据解释：

不能表示为现有特征的线性组合的“奇异特征向量”。

基：

对应于 0 奇异值的左奇异向量（$\boldsymbol{U}$ 中的其余列）。

列空间与行空间中的向量是能够被当前观测到的数据和特征表示出来的向量。列空间中的向量是非奇异的特征，行空间中的向量是非奇异的数据点。

对于建模和预测目的，非奇异是一件好事。一个满秩的列空间意味着特征集合中包含着足够的信息，可以对任何需要的目标向量进行建模。一个满秩的行空间意味着各个不同的数据点中包含着足够的变异，可以覆盖特征空间的各个角落。那些奇异的数据点和特征（分别位于零空间和左零空间中）才是我们需要担心的。

在为数据建立线性模型的应用中，零空间也可以看作“奇异”数据点的子空间。在这里，奇异性不是什么好事情。奇异的数据点表示那些不能被训练集线性表示的虚幻数据。同样，左零空间中包含的是不能表示为现有特征的线性组合的奇异特征。

零空间与行空间是正交的。原因很简单，从零空间的定义可知，$\boldsymbol{w}$ 与 $\boldsymbol{A}$ 中所有行向量的内积都是 0。因此，$\boldsymbol{w}$ 与这些行向量张成的空间（也就是行空间）是正交的。同样，左零空间与列空间也是正交的。

A.3 线性系统求解

让我们把这些数学知识应用到当前的问题中：训练线性分类器，这个问题与线性系统求解联系得非常紧密。我们深入研究了矩阵操作的原理，因为要对其进行逆向工程。为了训练一个线性分类器，我们必须找到权重输入向量 $\boldsymbol{w}$。在系统 $\boldsymbol{Aw} = \boldsymbol{y}$ 中，这个 $\boldsymbol{w}$ 能映射到观测

到的目标输出 $\boldsymbol{y}$，其中 $\boldsymbol{A}$ 是数据矩阵。[2]

下面逆向分析一下线性算子。如果已经有了 $\boldsymbol{A}$ 的奇异值分解，就可以将 $\boldsymbol{y}$ 映射到左奇异向量（$\boldsymbol{U}$ 的列）上，再逆转缩放因子（乘以非零奇异值的倒数），最后将它们映射回右奇异向量（$\boldsymbol{V}$ 的列）上。很简单，是不是？

这实际上是计算矩阵 $\boldsymbol{A}$ 的**伪逆**的过程。它利用了正交基的一个关键特性：转置就是逆矩阵。这就是 SVD 功能如此强大的原因。（实际上，真正的线性系统求解器不使用 SVD，因为它的计算成本太高。有成本更低的其他矩阵分解方法，比如 QR 分解、LU 分解、Cholesky 分解。）

但是，在我们的快速介绍中漏掉了一个小细节：如果奇异值为 0 应该怎么办呢？我们不能求 0 的倒数，因为 $1/0 = \infty$。这就是“伪逆”这个名称的由来。（真正的逆矩阵甚至不能定义在长方形矩阵上，只有特征值都不是 0 的方阵才有逆矩阵。）不管什么样的输入，都会被为 0 的奇异值粉碎得无影无踪，根本不能找回原来的输入。

好的，我们再把这个过程详细说一下，用所学的知识做一下线性系统的求解。假设我们得到了 $\boldsymbol{Aw} = \boldsymbol{y}$ 的一个答案，称其为 $\boldsymbol{w}_{\text{particular}}$，因为它特别适合 $\boldsymbol{y}$。假设还有一些能被 $\boldsymbol{A}$ 挤压为 0 的输入向量，我们取其中一个，称其为 $\boldsymbol{w}_{\text{sad-trumpet}}$，因为它哇哇大哭。然后，如果将 $\boldsymbol{w}_{\text{particular}}$ 和 $\boldsymbol{w}_{\text{sad-trumpet}}$ 相加，会发生什么呢？

$$\boldsymbol{A}(\boldsymbol{w}_{\text{particular}} + \boldsymbol{w}_{\text{sad-trumpet}}) = \boldsymbol{y}$$

太棒了！这也是一个解。实际上，任何能被挤压为 0 的输入都可以加到特定解上，从而再得到一个解。一般解可以表示如下：

$$\boldsymbol{w}_{\text{general}} = \boldsymbol{w}_{\text{particular}} + \boldsymbol{w}_{\text{homogeneous}}$$

$\boldsymbol{w}_{\text{particular}}$ 是方程 $\boldsymbol{Aw} = \boldsymbol{y}$ 的一个精确解，这个解可能存在，也可能不存在。如果没有精确解，那么线性系统只能近似求解。如果有精确解，那么 $\boldsymbol{y}$ 就属于 $\boldsymbol{A}$ 的列空间。列空间是 $\boldsymbol{A}$ 可以通过列的线性组合映射到的向量集合。

$\boldsymbol{w}_{\text{homogeneous}}$ 是方程 $\boldsymbol{Aw} = 0$ 的一个解。（$\boldsymbol{w}_{\text{sad-trumpet}}$ 长大之后，名字就变成了 $\boldsymbol{w}_{\text{homogeneous}}$。）这看上去似曾相识，所有 $\boldsymbol{w}_{\text{homogeneous}}$ 向量的集合构成了 $\boldsymbol{A}$ 的零空间，也就是奇异值为 0 的右奇异向量张成的空间。

注 2：实际上，情况还要更复杂一些。$\boldsymbol{y}$ 可能不在 $\boldsymbol{A}$ 的列空间中，所以这个问题可能没有解。这时，统计机器学习会寻找一个近似解，而不是放弃。它定义一个损失函数来对解的质量进行量化。如果解是精确的，损失函数的值就是 0。误差越小，损失函数的值越小；误差越大，损失函数的值越大，以此类推。这样，训练过程就是寻找最优参数来使损失函数最小化。在普通线性回归中，损失函数称为平方残差损失，它的本质作用就是将 $\boldsymbol{y}$ 映射到 $\boldsymbol{A}$ 的列空间中一个最接近的点。逻辑回归最小化的是对数损失。在这两种情形以及广义线性模型中，线性系统 $\boldsymbol{Aw} = \boldsymbol{y}$ 通常位于问题的核心。因此，这部分内容是非常重要的。

“零空间”这个名称听起来像是一种存在性危机的悲伤结局。如果零空间中除了全零向量之外还有其他向量，那么方程 $\boldsymbol{Aw} = \boldsymbol{y}$ 就有无穷多解。有很多解可供选择本身不是一件坏事，有时候我们可以选择任意一个解。但如果有很多可能的答案，就会有多个特征集合可以用于分类任务。这时就很难弄清楚哪个特征才是真正重要的。

解决零空间过大这个问题的一种方法是添加额外的限制条件，从而对模型进行**调整**：

$$\boldsymbol{Aw} = \boldsymbol{y}$$

其中 $\boldsymbol{w}$ 满足 $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} = c$。

这种正则化方式限制了权重向量具有特定的范数 c。这种正则化的强度是通过一个正则化参数来控制的，和前面实验中的做法一样，这个参数必须调优。

一般来说，**特征选择**方法需要选取出最有用的特征来降低计算负担，减轻模型的模糊性，并使得学习出的模型更与众不同。这也是 2.6 节中的重点。

另一个问题是数据矩阵的谱的“不均匀性”。在训练线性分类器时，我们关心的不仅是线性系统是否有通用解，还有我们是否能容易地找到这个解。通常，训练过程使用的求解器要计算损失函数的梯度，然后沿着梯度以较小的步长求解。当某些奇异值非常大而其余奇异值非常接近于 0 时，要想找到真实答案，求解器需要非常小心地绕过较长的奇异向量（对应于大奇异值的奇异向量），花费大量时间在较短的奇异向量附近进行探索。这种谱的“不均匀性”可以由矩阵的条件数来表示，也就是最大奇异值和最小奇异值的绝对值之间的比值。

总结一下，为了找出一个与众不同的好线性模型，也为了比较容易地找到它，我们需要以下条件。

(1) 标签向量能比较好地通过特征（列向量）的一个子集的线性组合进行近似，这个特征集合如果是线性独立的就更好了。
(2) 要使零空间较小，行空间必须很大。（这是因为这两个子空间是正交的。）数据点集合（行向量）越线性独立，零空间越小。
(3) 要想非常容易地找到解，数据矩阵的条件数（最大奇异值和最小奇异值之间的比值）应该较小。

A.4　参考文献

Strang, Gilbert. Linear Algebra and Its Applications [M]. 4th ed. Boston, MA: Cengage Learning, 2006.

作者简介

爱丽丝·郑（Alice Zheng）是应用机器学习、生成算法和平台开发领域的一位技术领导者。她现在担任 Amazon Advertising 的研究经理，此前曾在 GraphLab/Data/Turi 从事工具开发和用户培训，并在微软研究院担任过机器学习研究员。她在加州大学伯克利分校获得了计算机科学学士学位和数学学士学位以及电子工程和计算机科学博士学位。

阿曼达·卡萨丽（Amanda Casari）是一位领导者和工程师，研究兴趣是下一代技术以及如何充分展示出它们的影响。她现在是 Concur Labs 的高级产品经理和数据科学家，也是 SAP Concur 的 Concur Labs AI Research 团队的联合创始人。在过去的 16 年中，她担任过多种跨职能职务，涉足过多种工程领域，包括数据科学、机器学习、复杂系统和机器人。她在美国海军学院获得了控制系统工程学士学位，在佛蒙特大学获得了电子工程硕士学位。

封面简介

本书封面上的动物是一只法老雕鸮（学名 Bubo ascalaphus），这种食肉鸟类分布在北非和阿拉伯半岛干燥多岩石的地区。尽管雕鸮属中包括一些体型最大的猫头鹰，但法老雕鸮体型较小，体长 18~20 英寸（45~50 厘米）。多数雕鸮（和它们的美洲表亲角一样）都具有独一无二的耳羽。

法老雕鸮在夜间活动，站在高处进行捕食。它们居高临下，静候小型哺乳动物、蛇、蜥蜴、鸟类甚至昆虫进入捕猎范围，然后迅速地向猎物俯冲攻击。法老雕鸮的身体结构非常适合这种猎食方式，它们的听觉和视觉都非常敏锐，指爪坚硬而锐利，羽毛可以使它们飞行起来悄无声息，头可以在 270 度的范围内随意转动，这使得它们几乎不需要移动身体就能看到后方的情况。

这种动物具有褐色、黑色和白色相间的羽毛，以及与众不同的橙黄色眼睛。法老雕鸮以终生只有一个伴侣而著称，它们在岩石的凹处或裂缝之间选址筑巢，偶尔也会将巢搭在井架之类的人工建筑物上。在埃及，人们曾在金字塔上发现过法老雕鸮的巢穴。

O’Reilly 图书封面上的很多动物都是濒危物种，它们对这个世界都很重要。如果你想为保护这些动物做些贡献，请访问 animals.oreilly.com。

封面图片来自 *Elements of Ornithology*。

技术改变世界 · 阅读塑造人生

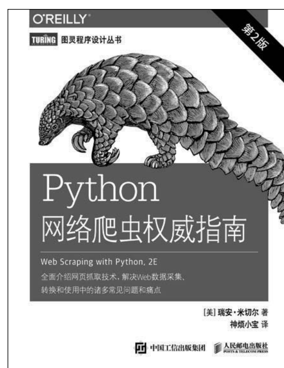


白话机器学习算法

- **斯坦福大学大数据基础课程教材**
- **文科生也看得懂的算法及数据科学入门书**
- **涵盖回归分析、神经网络、决策树、A/B测试等重要主题**

作者： 黄莉婷 苏川集

译者： 武传海

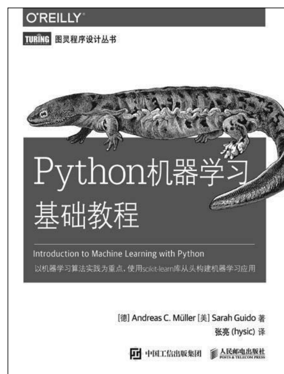


Python 网络爬虫权威指南（第 2 版）

- **全面介绍网页抓取技术，解决Web数据采集、转换和使用中的诸多常见问题和痛点**

作者： 瑞安・米切尔

译者： 神烦小宝



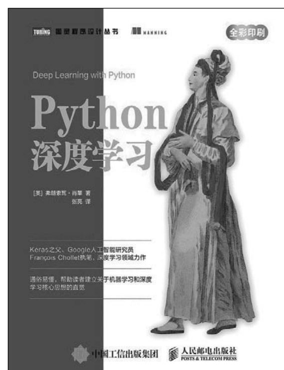
Python 机器学习基础教程

- **以机器学习算法实践为重点，使用scikit-learn库从头构建机器学习应用**

作者： Andreas C. Müller Sarah Guido

译者： 张亮（hysic）

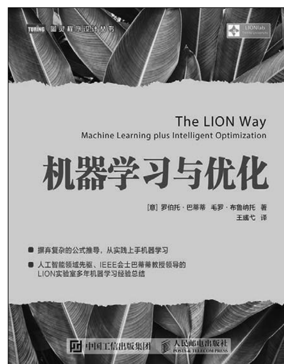
技术改变世界·阅读塑造人生



Python 深度学习

- Keras之父、Google人工智能研究员FrançoisChollet执笔，深度学习领域力作
- 通俗易懂，帮助读者建立关于机器学习和深度学习核心思想的直觉
- 16开全彩印刷

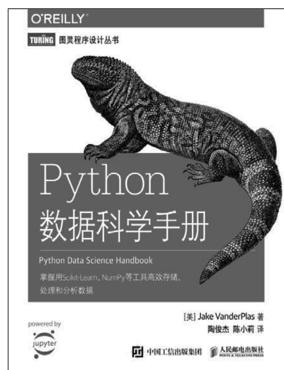
作者： 弗朗索瓦·肖莱
译者： 张亮



机器学习与优化

- 摒弃复杂的公式推导，从实践上手机器学习
- 人工智能领域先驱、IEEE会士巴蒂蒂教授领导的LION实验室多年机器学习经验总结

作者： 罗伯托·巴蒂蒂 毛罗·布鲁纳托
译者： 王彧弋



Python 数据科学手册

- 掌握用Scikit-Learn、NumPy等工具高效存储、处理和分析数据
- 大量示例+逐步讲解+举一反三，从计算环境配置到机器学习实战，切实解决工作痛点

作者： Jake VanderPlas
译者： 陶俊杰 陈小莉

O'REILLY®

精通特征工程

特征工程是机器学习流程中至关重要的一个环节，然而专门讨论这个话题的著作却寥寥无几。本书旨在填补这一空白，着重阐明特征工程的基本原则，介绍大量特征工程技术，教你从原始数据中提取出正确的特征并将其转换为适合机器学习模型的格式，从而轻松构建模型，增强机器学习算法的效果。

然而，本书并非单纯地讲述特征工程的基本原则，而是通过大量示例和练习将重点放在了实际应用上。每一章都集中研究一个数据问题：如何表示文本数据或图像数据，如何为自动生成的特征降低维度，何时以及如何对特征进行标准化，等等。最后一章通过一个完整的例子演示了多种特征工程技术的实际应用。书中所有代码示例均是用Python编写的，涉及NumPy、Pandas、scikit-learn和Matplotlib等程序包。

- 数值型数据的特征工程：过滤、分箱、缩放、对数变换和指数变换
- 自然文本技术：词袋、*n*元词与短语检测
- 基于频率的过滤和特征缩放
- 分类变量编码技术：特征散列化与分箱计数
- 使用主成分分析的基于模型的特征工程
- 模型堆叠与*k*-均值特征化
- 图像特征提取：人工提取与深度学习

“数据预处理和特征工程已成为很多应用中模型性能的决定因素。我十分欣喜地看到，终于有一本专门讨论这个话题的书了。Alice和Amanda条分缕析地介绍了多种常用技术之间的微妙差异。”

——Andreas C. Müller
scikit-learn库核心贡献者，
《Python机器学习基础教程》
合著者

爱丽丝·郑（Alice Zheng），亚马逊广告平台建模和优化团队负责人，应用机器学习、生成算法和平台开发领域的技术领导者，前微软研究院机器学习研究员。

阿曼达·卡萨丽（Amanda Casari），谷歌云开发者关系工程经理，曾是Concur Labs产品经理和数据科学家，在数据科学、机器学习、复杂系统和机器人等多个领域都有丰富经验。

封面设计：Karen Montgomery 张健

图灵社区：iTuring.cn
热线：(010)51095186转600



分类建议 计算机／机器学习

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)



ISBN 978-7-115-50968-0
定价：59.00元

看完了

如果您对本书内容有疑问，可发邮件至 contact@turingbook.com，会有编辑或作译者协助答疑。也可访问图灵社区，参与本书讨论。

如果是有关电子书的建议或问题，请联系专用客服邮箱:
ebook@turingbook.com。

在这可以找到我们:

微博 @图灵教育：好书、活动每日播报
微博 @图灵社区：电子书和好文章的消息
微博 @图灵新知：图灵教育的科普小组
微信 图灵访谈：ituring_interview，讲述码农精彩人生
微信 图灵教育：turingbooks